



Intel® 64 and IA-32 Architectures Software Developer's Manual

Documentation Changes

February 2026

Notice: The Intel® 64 and IA-32 architectures may contain design defects or errors known as errata that may cause the product to deviate from published specifications. Current characterized errata are documented in the specification updates.

Document Number: 252046-082



Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

All product plans and roadmaps are subject to change without notice.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that a) you may publish an unmodified copy and b) code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), <https://opensource.org/licenses/0BSD>. You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Contents

Revision History	4
Preface	9
Summary Tables of Changes	10
Documentation Changes	11



Revision History

Revision	Description	Date
-001	Initial release	November 2002
-002	- Added 1-10 Documentation Changes. - Removed old Documentation Changes items that already have been incorporated in the published Software Developer's manual	December 2002
-003	- Added 9 -17 Documentation Changes. - Removed Documentation Change #6 - References to bits Gen and Len Deleted. - Removed Documentation Change #4 - VIF Information Added to CLI Discussion	February 2003
-004	- Removed Documentation changes 1-17. - Added Documentation changes 1-24.	June 2003
-005	- Removed Documentation Changes 1-24. - Added Documentation Changes 1-15.	September 2003
-006	Added Documentation Changes 16- 34.	November 2003
-007	- Updated Documentation changes 14, 16, 17, and 28. - Added Documentation Changes 35-45.	January 2004
-008	- Removed Documentation Changes 1-45. - Added Documentation Changes 1-5.	March 2004
-009	Added Documentation Changes 7-27.	May 2004
-010	- Removed Documentation Changes 1-27. - Added Documentation Changes 1.	August 2004
-011	Added Documentation Changes 2-28.	November 2004
-012	- Removed Documentation Changes 1-28. - Added Documentation Changes 1-16.	March 2005
-013	- Updated title. - There are no Documentation Changes for this revision of the document.	July 2005
-014	Added Documentation Changes 1-21.	September 2005
-015	- Removed Documentation Changes 1-21. - Added Documentation Changes 1-20.	March 9, 2006
-016	Added Documentation changes 21-23.	March 27, 2006
-017	- Removed Documentation Changes 1-23. - Added Documentation Changes 1-36.	September 2006
-018	Added Documentation Changes 37-42.	October 2006
-019	- Removed Documentation Changes 1-42. - Added Documentation Changes 1-19.	March 2007
-020	Added Documentation Changes 20-27.	May 2007
-021	- Removed Documentation Changes 1-27. - Added Documentation Changes 1-6	November 2007
-022	- Removed Documentation Changes 1-6 - Added Documentation Changes 1-6	August 2008
-023	- Removed Documentation Changes 1-6 - Added Documentation Changes 1-21	March 2009

Revision	Description	Date
-024	- Removed Documentation Changes 1-21 - Added Documentation Changes 1-16	June 2009
-025	- Removed Documentation Changes 1-16 - Added Documentation Changes 1-18	September 2009
-026	- Removed Documentation Changes 1-18 - Added Documentation Changes 1-15	December 2009
-027	- Removed Documentation Changes 1-15 - Added Documentation Changes 1-24	March 2010
-028	- Removed Documentation Changes 1-24 - Added Documentation Changes 1-29	June 2010
-029	- Removed Documentation Changes 1-29 - Added Documentation Changes 1-29	September 2010
-030	- Removed Documentation Changes 1-29 - Added Documentation Changes 1-29	January 2011
-031	- Removed Documentation Changes 1-29 - Added Documentation Changes 1-29	April 2011
-032	- Removed Documentation Changes 1-29 - Added Documentation Changes 1-14	May 2011
-033	- Removed Documentation Changes 1-14 - Added Documentation Changes 1-38	October 2011
-034	- Removed Documentation Changes 1-38 - Added Documentation Changes 1-16	December 2011
-035	- Removed Documentation Changes 1-16 - Added Documentation Changes 1-18	March 2012
-036	- Removed Documentation Changes 1-18 - Added Documentation Changes 1-17	May 2012
-037	- Removed Documentation Changes 1-17 - Added Documentation Changes 1-28	August 2012
-038	- Removed Documentation Changes 1-28 - Add Documentation Changes 1-22	January 2013
-039	- Removed Documentation Changes 1-22 - Add Documentation Changes 1-17	June 2013
-040	- Removed Documentation Changes 1-17 - Add Documentation Changes 1-24	September 2013
-041	- Removed Documentation Changes 1-24 - Add Documentation Changes 1-20	February 2014
-042	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-8	February 2014
-043	- Removed Documentation Changes 1-8 - Add Documentation Changes 1-43	June 2014
-044	- Removed Documentation Changes 1-43 - Add Documentation Changes 1-12	September 2014
-045	- Removed Documentation Changes 1-12 - Add Documentation Changes 1-22	January 2015
-046	- Removed Documentation Changes 1-22 - Add Documentation Changes 1-25	April 2015
-047	- Removed Documentation Changes 1-25 - Add Documentation Changes 1-19	June 2015

Revision	Description	Date
-048	- Removed Documentation Changes 1-19 - Add Documentation Changes 1-33	September 2015
-049	- Removed Documentation Changes 1-33 - Add Documentation Changes 1-33	December 2015
-050	- Removed Documentation Changes 1-33 - Add Documentation Changes 1-9	April 2016
-051	- Removed Documentation Changes 1-9 - Add Documentation Changes 1-20	June 2016
-052	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-22	September 2016
-053	- Removed Documentation Changes 1-22 - Add Documentation Changes 1-26	December 2016
-054	- Removed Documentation Changes 1-26 - Add Documentation Changes 1-20	March 2017
-055	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-28	July 2017
-056	- Removed Documentation Changes 1-28 - Add Documentation Changes 1-18	October 2017
-057	- Removed Documentation Changes 1-18 - Add Documentation Changes 1-29	December 2017
-058	- Removed Documentation Changes 1-29 - Add Documentation Changes 1-17	March 2018
-059	- Removed Documentation Changes 1-17 - Add Documentation Changes 1-24	May 2018
-060	- Removed Documentation Changes 1-24 - Add Documentation Changes 1-23	November 2018
-061	- Removed Documentation Changes 1-23 - Add Documentation Changes 1-21	January 2019
-062	- Removed Documentation Changes 1-21 - Add Documentation Changes 1-28	May 2019
-063	- Removed Documentation Changes 1-28 - Add Documentation Changes 1-34	October 2019
-064	- Removed Documentation Changes 1-34 - Add Documentation Changes 1-36	May 2020
-065	- Removed Documentation Changes 1-36 - Add Documentation Changes 1-31	November 2020
-066	- Removed Documentation Changes 1-31 - Add Documentation Changes 1-24	April 2021
-067	- Removed Documentation Changes 1-24 - Add Documentation Changes 1-30	June 2021
-068	- Removed Documentation Changes 1-30 - Add Documentation Changes 1-29	December 2021
-069	- Removed Documentation Changes 1-29 - Add Documentation Changes 1-18	April 2022
-070	- Removed Documentation Changes 1-18 - Add Documentation Changes 1-41	December 2022
-071	- Removed Documentation Changes 1-41 - Add Documentation Changes 1-23	March 2023

Revision	Description	Date
-072	- Removed Documentation Changes 1-23 - Add Documentation Changes 1-19	June 2023
-073	- Removed Documentation Changes 1-19 - Add Documentation Changes 1-19	September 2023
-074	- Removed Documentation Changes 1-19 - Add Documentation Changes 1-20	December 2023
-075	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-20	March 2024
-076	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-8	June 2024
-077	- Removed Documentation Changes 1-8 - Add Documentation Changes 1-27	October 2024
-078	- Removed Documentation Changes 1-27 - Add Documentation Changes 1-15	December 2024
-079	- Removed Documentation Changes 1-15 - Add Documentation Changes 1-17	March 2025
-080	- Removed Documentation Changes 1-17 - Add Documentation Changes 1-25	June 2025
-081	- Removed Documentation Changes 1-25 - Add Documentation Changes 1-33	October 2025
-082	- Removed Documentation Changes 1-33 - Added Documentation Changes 1-32	January 2026

§



Preface

This document is an update to the specifications contained in the [Affected Documents](#) table below. This document is a compilation of device and documentation errata, specification clarifications and changes. It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools.

Affected Documents

Document Title	Document Number/ Location
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture</i>	253665
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-L</i>	253666
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, M-U</i>	253667
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C: Instruction Set Reference, V</i>	326018
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D: Instruction Set Reference, W-Z</i>	334569
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1</i>	253668
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2</i>	253669
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3</i>	326019
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4</i>	332831
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model Specific Registers</i>	335592

Nomenclature

Documentation Changes include typos, errors, or omissions from the current published specifications. These will be incorporated in any new release of the specification.

Summary Tables of Changes

The following table indicates documentation changes which apply to the Intel® 64 and IA-32 architectures. This table uses the following notations:

Codes Used in Summary Tables

A change bar to left of table row indicates this erratum is either new or modified from the previous version of the document.

Documentation Changes(Sheet 1 of 2)

No.	DOCUMENTATION CHANGES
1	Updates to Chapter 5, Volume 1
2	Updates to Chapter 6, Volume 1
3	Updates to Chapter 7, Volume 1
4	Updates to Chapter 18, Volume 1
5	Updates to Chapter 21, Volume 1
6	Updates to Chapter 3, Volume 2A
7	Updates to Chapter 4, Volume 2B
8	Updates to Chapter 5, Volume 2C
9	Updates to Chapter 7, Volume 2D
10	Updates to Appendix A, Volume 2D
11	Updates to Chapter 2, Volume 3A
12	Updates to Chapter 6, Volume 3A
13	Updates to Chapter 7, Volume 3A
14	Updates to Chapter 8, Volume 3A
15	Updates to Chapter 9, Volume 3A
16	Updates to Chapter 11, Volume 3A
17	Updates to Chapter 12, Volume 3A
18	Updates to Chapter 14, Volume 3A
19	Updates to Chapter 20, Volume 3B
20	Updates to Chapter 22, Volume 3B
21	Updates to Chapter 27, Volume 3C
22	Updates to Chapter 28, Volume 3C
23	Updates to Chapter 29, Volume 3C
24	Updates to Chapter 30, Volume 3C
25	Updates to Chapter 32, Volume 3C
26	Updates to Chapter 34, Volume 3C

Documentation Changes(Sheet 2 of 2)

No.	DOCUMENTATION CHANGES
27	Updates to Chapter 35, Volume 3C
28	Updates to Chapter 36, Volume 3D
29	Updates to Chapter 37, Volume 3D
30	Updates Appendix A, Volume 3D
31	Updates Appendix B, Volume 3D
32	Updates to Chapter 2, Volume 4

Documentation Changes

Changes to the Intel® 64 and IA-32 Architectures Software Developer's Manual volumes follow, and are listed by chapter. Only chapters with changes are included in this document.

1. Updates to Chapter 5, Volume 1

Change bars and text show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

Changes to this chapter:

- In Section 5.19, "Intel® Advanced Vector Extensions 512 (Intel® AVX-512)," the list of mnemonics in AVX-512BW that are not Intel AVX or AVX2 was updated for VPERMI2W.

This chapter provides an abridged overview of Intel 64 and IA-32 instructions. Instructions are divided into the following groups:

- Section 5.1, "General-Purpose Instructions."
- Section 5.2, "x87 FPU Instructions."
- Section 5.3, "x87 FPU AND SIMD State Management Instructions."
- Section 5.4, "MMX Instructions."
- Section 5.5, "Intel® SSE Instructions."
- Section 5.6, "Intel® SSE2 Instructions."
- Section 5.7, "Intel® SSE3 Instructions."
- Section 5.8, "Supplemental Streaming SIMD Extensions 3 (SSSE3) Instructions."
- Section 5.9, "Intel® SSE4 Instructions."
- Section 5.10, "Intel® SSE4.1 Instructions."
- Section 5.11, "Intel® SSE4.2 Instruction Set."
- Section 5.12, "Intel® AES-NI and PCLMULQDQ."
- Section 5.13, "Intel® Advanced Vector Extensions (Intel® AVX)."
- Section 5.14, "16-bit Floating-Point Conversion."
- Section 5.15, "Fused-Multiply-ADD (FMA)."
- Section 5.16, "Intel® Advanced Vector Extensions 2 (Intel® AVX2)."
- Section 5.17, "Intel® Transactional Synchronization Extensions (Intel® TSX)."
- Section 5.18, "Intel® SHA Extensions."
- Section 5.19, "Intel® Advanced Vector Extensions 512 (Intel® AVX-512)."
- Section 5.20, "System Instructions."
- Section 5.21, "64-Bit Mode Instructions."
- Section 5.22, "Virtual-Machine Extensions."
- Section 5.23, "Safer Mode Extensions."
- Section 5.24, "Intel® Memory Protection Extensions."
- Section 5.25, "Intel® Software Guard Extensions."
- Section 5.26, "Shadow Stack Management Instructions."
- Section 5.27, "Control Transfer Terminating Instructions."
- Section 5.28, "Intel® AMX Instructions."
- Section 5.29, "User Interrupt Instructions."
- Section 5.30, "Enqueue Store Instructions."
- Section 5.31, "Intel® Advanced Vector Extensions 10 Version 1 Instructions."

Table 5-1 lists the groups and IA-32 processors that support each group. More recent instruction set extensions are listed in Table 5-2. Within these groups, most instructions are collected into functional subgroups.

Table 5-1. Instruction Groups in Intel® 64 and IA-32 Processors

Instruction Set Architecture	Intel 64 and IA-32 Processor Support
General Purpose	All Intel 64 and IA-32 processors.
X87 FPU	Intel486, Pentium, Pentium with MMX Technology, Celeron, Pentium Pro, Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors, Pentium M, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Atom processors.
X87 FPU and SIMD State Management	Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors, Pentium M, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Atom processors.
MMX Technology	Pentium with MMX Technology, Celeron, Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors, Pentium M, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Atom processors.
SSE Extensions	Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors, Pentium M, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Atom processors.
SSE2 Extensions	Pentium 4, Intel Xeon processors, Pentium M, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Atom processors.
SSE3 Extensions	Pentium 4 supporting HT Technology (built on 90 nm process technology), Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Xeon processor 3xxx, 5xxx, 7xxx Series, Intel Atom processors.
SSSE3 Extensions	Intel Xeon processor 3xxx, 5100, 5200, 5300, 5400, 5500, 5600, 7300, 7400, 7500 series, Intel Core 2 Extreme processors QX6000 series, Intel Core 2 Duo, Intel Core 2 Quad processors, Intel Pentium Dual-Core processors, Intel Atom processors.
IA-32e mode: 64-bit mode instructions	Intel 64 processors.
System Instructions	Intel 64 and IA-32 processors.
VMX Instructions	Intel 64 and IA-32 processors supporting Intel Virtualization Technology.
SMX Instructions	Intel Core 2 Duo processor E6x50, E8xxx; Intel Core 2 Quad processor Q9xxx.

Table 5-2. Instruction Set Extensions Introduction in Intel® 64 and IA-32 Processors

Instruction Set Architecture	Processor Generation Introduction
SSE4.1 Extensions	Intel® Xeon® processor 3100, 3300, 5200, 5400, 7400, 7500 series, Intel® Core™ 2 Extreme processors QX9000 series, Intel® Core™ 2 Quad processor Q9000 series, Intel® Core™ 2 Duo processors 8000 series and T9000 series, Intel Atom® processor based on Silvermont microarchitecture.
SSE4.2 Extensions, CRC32, POPCNT	Intel® Core™ i7 965 processor, Intel® Xeon® processors X3400, X3500, X5500, X6500, X7500 series, Intel Atom processor based on Silvermont microarchitecture.
Intel® AES-NI, PCLMULQDQ	Intel® Xeon® processor E7 series, Intel® Xeon® processors X3600 and X5600, Intel® Core™ i7 980X processor, Intel Atom processor based on Silvermont microarchitecture. Use CPUID to verify presence of Intel AES-NI and PCLMULQDQ across Intel® Core™ processor families.
Intel® AVX	Intel® Xeon® processor E3 and E5 families, 2nd Generation Intel® Core™ i7, i5, i3 processor 2xxx families.
F16C	3rd Generation Intel® Core™ processors, Intel® Xeon® processor E3-1200 v2 product family, Intel® Xeon® processor E5 v2 and E7 v2 families.
RDRAND	3rd Generation Intel Core processors, Intel Xeon processor E3-1200 v2 product family, Intel Xeon processor E5 v2 and E7 v2 families, Intel Atom processor based on Silvermont microarchitecture.
FS/GS base access	3rd Generation Intel Core processors, Intel Xeon processor E3-1200 v2 product family, Intel Xeon processor E5 v2 and E7 v2 families, Intel Atom® processor based on Goldmont microarchitecture.

Table 5-2. Instruction Set Extensions Introduction in Intel® 64 and IA-32 Processors (Contd.)

Instruction Set Architecture	Processor Generation Introduction
FMA, AVX2, BMI1, BMI2, INVPCID, LZCNT, Intel® TSX	Intel® Xeon® processor E3/E5/E7 v3 product families, 4th Generation Intel® Core™ processor family.
MOVBE	Intel Xeon processor E3/E5/E7 v3 product families, 4th Generation Intel Core processor family, Intel Atom processors.
PREFETCHW	Intel® Core™ M processor family; 5th Generation Intel® Core™ processor family, Intel Atom processor based on Silvermont microarchitecture.
ADX	Intel Core M processor family, 5th Generation Intel Core processor family.
RDSEED, CLAC, STAC	Intel Core M processor family, 5th Generation Intel Core processor family, Intel Atom processor based on Goldmont microarchitecture.
AVX512ER, AVX512PF, PREFETCHWT1	Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series.
AVX512F, AVX512CD	Intel Xeon Phi Processor 3200, 5200, 7200 Series, Intel® Xeon® Scalable Processor Family, Intel® Core™ i3-8121U processor.
CLFLUSHOPT, XSAVEC, XSAVES, Intel® MPX	Intel Xeon Scalable Processor Family, 6th Generation Intel® Core™ processor family, Intel Atom processor based on Goldmont microarchitecture.
SGX1	6th Generation Intel Core processor family, Intel Atom® processor based on Goldmont Plus microarchitecture.
AVX512DQ, AVX512BW, AVX512VL	Intel Xeon Scalable Processor Family, Intel Core i3-8121U processor based on Cannon Lake microarchitecture.
CLWB	Intel Xeon Scalable Processor Family, Intel Atom® processor based on Tremont microarchitecture, 11th Generation Intel Core processor family based on Tiger Lake microarchitecture.
PKU	Intel Xeon Scalable Processor Family, 10th generation Intel® Core™ processors based on Comet Lake microarchitecture.
AVX512_IFMA, AVX512_VBMI	Intel Core i3-8121U processor based on Cannon Lake microarchitecture.
Intel® SHA Extensions	Intel Core i3-8121U processor based on Cannon Lake microarchitecture, Intel Atom processor based on Goldmont microarchitecture, 3rd Generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture.
UMIP	Intel Core i3-8121U processor based on Cannon Lake microarchitecture, Intel Atom processor based on Goldmont Plus microarchitecture.
PTWRITE	Intel Atom processor based on Goldmont Plus microarchitecture, 12th generation Intel® Core™ processor supporting Alder Lake performance hybrid architecture, 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture.
RDPID	10th Generation Intel® Core™ processor family based on Ice Lake microarchitecture, Intel Atom processor based on Goldmont Plus microarchitecture.
AVX512_4FMAPS, AVX512_4VNNIW	Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series.
AVX512_VNNI	2nd Generation Intel® Xeon® Scalable Processor Family, 10th Generation Intel Core processor family based on Ice Lake microarchitecture.
AVX512_VPOPCNTDQ	Intel Xeon Phi Processor 7215, 7285, 7295 Series, 10th Generation Intel Core processor family based on Ice Lake microarchitecture.
Fast Short REP MOVSB	10th Generation Intel Core processor family based on Ice Lake microarchitecture.
GFNI (SSE)	10th Generation Intel Core processor family based on Ice Lake microarchitecture, Intel Atom processor based on Tremont microarchitecture.

Table 5-2. Instruction Set Extensions Introduction in Intel® 64 and IA-32 Processors (Contd.)

Instruction Set Architecture	Processor Generation Introduction
VAES, GFNI (AVX/AVX512), AVX512_VBMI2, VPCLMULQDQ, AVX512_BITALG	10th Generation Intel Core processor family based on Ice Lake microarchitecture.
Split Lock Detection	10th Generation Intel Core processor family based on Ice Lake microarchitecture, Intel Atom processor based on Tremont microarchitecture.
CLDEMOT	Intel Atom processor based on Tremont microarchitecture, 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture.
Direct stores: MOVDIRI, MOVDIR64B	Intel Atom processor based on Tremont microarchitecture, 11th Generation Intel Core processor family based on Tiger Lake microarchitecture, 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture.
User wait: TPAUSE, UMONITOR, UMWAIT	Intel Atom processor based on Tremont microarchitecture, 12th generation Intel Core processor based on Alder Lake performance hybrid architecture, 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture.
AVX512-BF16	3rd Generation Intel® Xeon® Scalable Processor Family based on Cooper Lake product, 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture.
AVX512_VP2INTERSECT	11th Generation Intel Core processor family based on Tiger Lake microarchitecture. (Not currently supported in any other processors).
Key Locker ¹	11th Generation Intel Core processor family based on Tiger Lake microarchitecture, 12th generation Intel Core processor supporting Alder Lake performance hybrid architecture.
Control-flow Enforcement Technology (CET)	11th Generation Intel Core processor family based on Tiger Lake microarchitecture, 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture, Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture.
TME-MK ² , PCONFIG	3rd Generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture.
WBNOINVD	3rd Generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture.
LBRs (architectural)	12th generation Intel Core processor supporting Alder Lake performance hybrid architecture, 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture, Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture.
Intel® Virtualization Technology - Redirect Protection (Intel® VT-rp) and HLAT	12th generation Intel Core processor supporting Alder Lake performance hybrid architecture, 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture, Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture.
AVX-VNNI	12th generation Intel Core processor supporting Alder Lake performance hybrid architecture ³ , 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture, Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture.
SERIALIZE	12th generation Intel Core processor supporting Alder Lake performance hybrid architecture, 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture, Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture.
Intel® Thread Director and HRESET	12th generation Intel Core processor supporting Alder Lake performance hybrid architecture.
Fast zero-length REP MOVSB, fast short REP STOSB	12th generation Intel Core processor supporting Alder Lake performance hybrid architecture, 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture.
Fast Short REP CMPSB, fast short REP SCASB	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture.
Supervisor Memory Protection Keys (PKS)	12th generation Intel Core processor supporting Alder Lake performance hybrid architecture, 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture, Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture.

Table 5-2. Instruction Set Extensions Introduction in Intel® 64 and IA-32 Processors (Contd.)

Instruction Set Architecture	Processor Generation Introduction
Attestation Services for Intel® SGX	3rd Generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture.
Enqueue Stores: ENQCMD and ENQCMLS	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture, Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture.
Intel® TSX Suspend Load Address Tracking (TSXLDTRK)	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture.
Intel® Advanced Matrix Extensions (Intel® AMX) Includes CPUID.1EH, “TMUL Information Main Leaf”, and CPUID bits AMX_BF16, AMX_TILE, and AMX_INT8.	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture.
User Interrupts (UINTR)	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture, Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture, Intel® Core™ Ultra 200H Series processors supporting Arrow Lake performance hybrid architecture.
IPI Virtualization	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture, Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture, Intel® Core™ Ultra 200H Series processors supporting Arrow Lake performance hybrid architecture.
AVX512-FP16 for the FP16 Data Type	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture.
Virtualization of guest accesses to IA32_SPEC_CTRL	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture, Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture.
Linear Address Masking (LAM)	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture, Intel® Core™ Ultra 200H Series processors supporting Arrow Lake performance hybrid architecture.
Linear Address Space Separation (LASS)	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture, Intel® Core™ Ultra 200H Series processors supporting Arrow Lake performance hybrid architecture.
PREFETCHIT0/1	Intel® Xeon® 6 P-core processors based on Granite Rapids microarchitecture.
AMX_FP16	Intel® Xeon® 6 P-core processors based on Granite Rapids microarchitecture.
CMPCCXADD	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture, Intel® Core™ Ultra 200H Series processors supporting Arrow Lake performance hybrid architecture.
AVX-IFMA	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture, Intel® Core™ Ultra 200H Series processors supporting Arrow Lake performance hybrid architecture.
AVX-NE-CONVERT	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture, Intel® Core™ Ultra 200H Series processors supporting Arrow Lake performance hybrid architecture.
AVX-VNNI-INT8	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture, Intel® Core™ Ultra 200H Series processors supporting Arrow Lake performance hybrid architecture.
AVX-VNNI-INT16	Intel® Core™ Ultra 200S Series processors supporting Arrow Lake performance hybrid architecture.
SHA512	Intel® Core™ Ultra 200S Series processors supporting Arrow Lake performance hybrid architecture.

Table 5-2. Instruction Set Extensions Introduction in Intel® 64 and IA-32 Processors (Contd.)

Instruction Set Architecture	Processor Generation Introduction
SM3	Intel® Core™ Ultra 200S Series processors supporting Arrow Lake performance hybrid architecture.
SM4	Intel® Core™ Ultra 200S Series processors supporting Arrow Lake performance hybrid architecture.
RDMSRLIST, WRMSRLIST, and WRMSRNS	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture.
UC Lock Disable Causes #AC	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture.
LBR Event Logging	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture, Intel® Core™ Ultra 200S Series processors supporting Arrow Lake performance hybrid architecture.
UIRET flexibly updates UIF	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture, Intel® Atom® P6900 Processor based on the Crestmont microarchitecture, Intel® Core™ Ultra 200H Series processors supporting Arrow Lake performance hybrid architecture.
Intel® Advanced Vector Extensions 10 Version 1 (Intel® AVX10.1)	Intel® Xeon® 6 P-core processors based on Granite Rapids microarchitecture.

NOTES:

1. Details on Key Locker can be found in the Intel Key Locker Specification here:
<https://software.intel.com/content/www/us/en/develop/download/intel-key-locker-specification.html>.
2. Further details on TME-MK usage can be found here:
<https://software.intel.com/sites/default/files/managed/a5/16/Multi-Key-Total-Memory-Encryption-Spec.pdf>.
3. Alder Lake performance hybrid architecture does not support Intel® AVX-512. ISA features such as Intel® AVX, AVX-VNNI, Intel® AVX2, and UMONITOR/UMWAIT/TPAUSE are supported.

The following sections list instructions in each major group and subgroup. Given for each instruction is its mnemonic and descriptive names. When two or more mnemonics are given (for example, CMOVA/CMOVNBE), they represent different mnemonics for the same instruction opcode. Assemblers support redundant mnemonics for some instructions to make it easier to read code listings. For instance, CMOVA (Conditional move if above) and CMOVNBE (Conditional move if not below or equal) represent the same condition. For detailed information about specific instructions, see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D.

5.1 GENERAL-PURPOSE INSTRUCTIONS

The general-purpose instructions perform basic data movement, arithmetic, logic, program flow, and string operations that programmers commonly use to write application and system software to run on Intel 64 and IA-32 processors. They operate on data contained in memory, in the general-purpose registers (EAX, EBX, ECX, EDX, EDI, ESI, EBP, and ESP) and in the EFLAGS register. They also operate on address information contained in memory, the general-purpose registers, and the segment registers (CS, DS, SS, ES, FS, and GS).

This group of instructions includes the data transfer, binary integer arithmetic, decimal arithmetic, logic operations, shift and rotate, bit and byte operations, program control, string, flag control, segment register operations, and miscellaneous subgroups. The sections that follow introduce each subgroup.

For more detailed information on general purpose-instructions, see Chapter 7, "Programming With General-Purpose Instructions."

5.1.1 Data Transfer Instructions

The data transfer instructions move data between memory and the general-purpose and segment registers. They also perform specific operations such as conditional moves, stack access, and data conversion.

MOV	Move data between general-purpose registers; move data between memory and general-purpose or segment registers; move immediates to general-purpose registers.
CMOVE/CMOVZ	Conditional move if equal/Conditional move if zero.
CMOVNE/CMOVNZ	Conditional move if not equal/Conditional move if not zero.
CMOVA/CMOVNBE	Conditional move if above/Conditional move if not below or equal.
CMOVAE/CMOVNB	Conditional move if above or equal/Conditional move if not below.
CMOVNB/CMOVNAE	Conditional move if below/Conditional move if not above or equal.
CMOVBE/CMOVNA	Conditional move if below or equal/Conditional move if not above.
CMOVG/CMOVNLE	Conditional move if greater/Conditional move if not less or equal.
CMOVGE/CMOVNL	Conditional move if greater or equal/Conditional move if not less.
CMOVL/CMOVNGE	Conditional move if less/Conditional move if not greater or equal.
CMOVLE/CMOVNG	Conditional move if less or equal/Conditional move if not greater.
CMOVC	Conditional move if carry.
CMOVNC	Conditional move if not carry.
CMOVO	Conditional move if overflow.
CMOVNO	Conditional move if not overflow.
CMOVS	Conditional move if sign (negative).
CMOVNS	Conditional move if not sign (non-negative).
CMOVP/CMOVPE	Conditional move if parity/Conditional move if parity even.
CMOVNP/CMOVPO	Conditional move if not parity/Conditional move if parity odd.
XCHG	Exchange.
BSWAP	Byte swap.
XADD	Exchange and add.
CMPXCHG	Compare and exchange.
CMPXCHG8B	Compare and exchange 8 bytes.
PUSH	Push onto stack.
POP	Pop off of stack.
PUSHA/PUSHAD	Push general-purpose registers onto stack.
POPA/POPAD	Pop general-purpose registers from stack.
CWD/CDQ	Convert word to doubleword/Convert doubleword to quadword.
CBW/CWDE	Convert byte to word/Convert word to doubleword in EAX register.
MOVSX	Move and sign extend.
MOVZX	Move and zero extend.

5.1.2 Binary Arithmetic Instructions

The binary arithmetic instructions perform basic binary integer computations on byte, word, and doubleword integers located in memory and/or the general purpose registers.

ADCX	Unsigned integer add with carry.
ADOX	Unsigned integer add with overflow.
ADD	Integer add.
ADC	Add with carry.
SUB	Subtract.
SBB	Subtract with borrow.

IMUL	Signed multiply.
MUL	Unsigned multiply.
IDIV	Signed divide.
DIV	Unsigned divide.
INC	Increment.
DEC	Decrement.
NEG	Negate.
CMP	Compare.

5.1.3 Decimal Arithmetic Instructions

The decimal arithmetic instructions perform decimal arithmetic on binary coded decimal (BCD) data.

DAA	Decimal adjust after addition.
DAS	Decimal adjust after subtraction.
AAA	ASCII adjust after addition.
AAS	ASCII adjust after subtraction.
AAM	ASCII adjust after multiplication.
AAD	ASCII adjust before division.

5.1.4 Logical Instructions

The logical instructions perform basic AND, OR, XOR, and NOT logical operations on byte, word, and doubleword values.

AND	Perform bitwise logical AND.
OR	Perform bitwise logical OR.
XOR	Perform bitwise logical exclusive OR.
NOT	Perform bitwise logical NOT.

5.1.5 Shift and Rotate Instructions

The shift and rotate instructions shift and rotate the bits in word and doubleword operands.

SAR	Shift arithmetic right.
SHR	Shift logical right.
SAL/SHL	Shift arithmetic left/Shift logical left.
SHRD	Shift right double.
SHLD	Shift left double.
ROR	Rotate right.
ROL	Rotate left.
RCR	Rotate through carry right.
RCL	Rotate through carry left.

5.1.6 Bit and Byte Instructions

Bit instructions test and modify individual bits in word and doubleword operands. Byte instructions set the value of a byte operand to indicate the status of flags in the EFLAGS register.

BT	Bit test.
BTS	Bit test and set.

BTR	Bit test and reset.
BTC	Bit test and complement.
BSF	Bit scan forward.
BSR	Bit scan reverse.
SETE/SETZ	Set byte if equal/Set byte if zero.
SETNE/SETNZ	Set byte if not equal/Set byte if not zero.
SETA/SETNBE	Set byte if above/Set byte if not below or equal.
SETAE/SETNB/SETNC	Set byte if above or equal/Set byte if not below/Set byte if not carry.
SETB/SETNAE/SETC	Set byte if below/Set byte if not above or equal/Set byte if carry.
SETBE/SETNA	Set byte if below or equal/Set byte if not above.
SETG/SETNLE	Set byte if greater/Set byte if not less or equal.
SETGE/SETNL	Set byte if greater or equal/Set byte if not less.
SETL/SETNGE	Set byte if less/Set byte if not greater or equal.
SETLE/SETNG	Set byte if less or equal/Set byte if not greater.
SETS	Set byte if sign (negative).
SETNS	Set byte if not sign (non-negative).
SETO	Set byte if overflow.
SETNO	Set byte if not overflow.
SETPE/SETP	Set byte if parity even/Set byte if parity.
SETPO/SETNP	Set byte if parity odd/Set byte if not parity.
TEST	Logical compare.
CRC32 ¹	Provides hardware acceleration to calculate cyclic redundancy checks for fast and efficient implementation of data integrity protocols.
POPCNT ²	Calculates of number of bits set to 1 in the second operand (source) and returns the count in the first operand (a destination register).

5.1.7 Control Transfer Instructions

The control transfer instructions provide jump, conditional jump, loop, and call and return operations to control program flow.

JMP	Jump.
JE/JZ	Jump if equal/Jump if zero.
JNE/JNZ	Jump if not equal/Jump if not zero.
JA/JNBE	Jump if above/Jump if not below or equal.
JAЕ/JNB	Jump if above or equal/Jump if not below.
JB/JNAE	Jump if below/Jump if not above or equal.
JBE/JNA	Jump if below or equal/Jump if not above.
JG/JNLE	Jump if greater/Jump if not less or equal.
JGE/JNL	Jump if greater or equal/Jump if not less.
JL/JNGE	Jump if less/Jump if not greater or equal.
JLE/JNG	Jump if less or equal/Jump if not greater.
JC	Jump if carry.
JNC	Jump if not carry.
JO	Jump if overflow.

1. Processor support of CRC32 is enumerated by CPUID.01H:ECX.SSE4_2 = 1

2. Processor support of POPCNT is enumerated by CPUID.01H:ECX.POPCNT = 1

JNO	Jump if not overflow.
JS	Jump if sign (negative).
JNS	Jump if not sign (non-negative).
JPO/JNP	Jump if parity odd/Jump if not parity.
JPE/JP	Jump if parity even/Jump if parity.
JCXZ/JECXZ	Jump register CX zero/Jump register ECX zero.
LOOP	Loop with ECX counter.
LOOPZ/LOOPE	Loop with ECX and zero/Loop with ECX and equal.
LOOPNZ/LOOPNE	Loop with ECX and not zero/Loop with ECX and not equal.
CALL	Call procedure.
RET	Return.
IRET	Return from interrupt.
INT	Software interrupt.
INTO	Interrupt on overflow.
BOUND	Detect value out of range.
ENTER	High-level procedure entry.
LEAVE	High-level procedure exit.

5.1.8 String Instructions

The string instructions operate on strings of bytes, allowing them to be moved to and from memory.

MOVS/MOVS	Move string/Move byte string.
MOVS/MOVS	Move string/Move word string.
MOVS/MOVS	Move string/Move doubleword string.
CMPS/CMPS	Compare string/Compare byte string.
CMPS/CMPS	Compare string/Compare word string.
CMPS/CMPS	Compare string/Compare doubleword string.
SCAS/SCAS	Scan string/Scan byte string.
SCAS/SCAS	Scan string/Scan word string.
SCAS/SCAS	Scan string/Scan doubleword string.
LODS/LODS	Load string/Load byte string.
LODS/LODS	Load string/Load word string.
LODS/LODS	Load string/Load doubleword string.
STOS/STOS	Store string/Store byte string.
STOS/STOS	Store string/Store word string.
STOS/STOS	Store string/Store doubleword string.
REP	Repeat while ECX not zero.
REPE/REPZ	Repeat while equal/Repeat while zero.
REPNE/REPZ	Repeat while not equal/Repeat while not zero.

5.1.9 I/O Instructions

These instructions move data between the processor's I/O ports and a register or memory.

IN	Read from a port.
OUT	Write to a port.
INS/INSB	Input string from port/Input byte string from port.
INS/INSW	Input string from port/Input word string from port.

INS/INSD	Input string from port/Input doubleword string from port.
OUTS/OUTSB	Output string to port/Output byte string to port.
OUTS/OUTSW	Output string to port/Output word string to port.
OUTS/OUTSD	Output string to port/Output doubleword string to port.

5.1.10 Enter and Leave Instructions

These instructions provide machine-language support for procedure calls in block-structured languages.

ENTER	High-level procedure entry.
LEAVE	High-level procedure exit.

5.1.11 Flag Control (EFLAG) Instructions

The flag control instructions operate on the flags in the EFLAGS register.

STC	Set carry flag.
CLC	Clear the carry flag.
CMC	Complement the carry flag.
CLD	Clear the direction flag.
STD	Set direction flag.
LAHF	Load flags into AH register.
SAHF	Store AH register into flags.
PUSHF/PUSHFD	Push EFLAGS onto stack.
POPF/POPF	Pop EFLAGS from stack.
STI	Set interrupt flag.
CLI	Clear the interrupt flag.

5.1.12 Segment Register Instructions

The segment register instructions allow far pointers (segment addresses) to be loaded into the segment registers.

LDS	Load far pointer using DS.
LES	Load far pointer using ES.
LFS	Load far pointer using FS.
LGS	Load far pointer using GS.
LSS	Load far pointer using SS.

5.1.13 Miscellaneous Instructions

The miscellaneous instructions provide such functions as loading an effective address, executing a “no-operation,” and retrieving processor identification information.

LEA	Load effective address.
NOP	No operation.
UD	Undefined instruction.
XLAT/XLATB	Table lookup translation.
CPUID	Processor identification.
MOVBE ¹	Move data after swapping data bytes.

1. Processor support of MOVBE is enumerated by CPUID.01H:ECX.MOVBE[22] = 1.

PREFETCHW	Prefetch data into cache in anticipation of write.
PREFETCHWT1	Prefetch hint T1 with intent to write.
CLFLUSH	Flushes and invalidates a memory operand and its associated cache line from all levels of the processor's cache hierarchy.
CLFLUSHOPT	Flushes and invalidates a memory operand and its associated cache line from all levels of the processor's cache hierarchy with optimized memory system throughput.

5.1.14 User Mode Extended State Save/Restore Instructions

XSAVE	Save processor extended states to memory.
XSAVEC	Save processor extended states with compaction to memory.
XSAVEOPT	Save processor extended states to memory, optimized.
XRSTOR	Restore processor extended states from memory.
XGETBV	Reads the state of an extended control register.

5.1.15 Random Number Generator Instructions

RDRAND	Retrieves a random number generated from hardware.
RDSEED	Retrieves a random number generated from hardware.

5.1.16 BMI1 and BMI2 Instructions

ANDN	Bitwise AND of first source with inverted second source operands.
BEXTR	Contiguous bitwise extract.
BLSI	Extract lowest set bit.
BLSMSK	Set all lower bits below first set bit to 1.
BLSR	Reset lowest set bit.
BZHI	Zero high bits starting from specified bit position.
LZCNT	Count the number of leading zero bits.
MULX	Unsigned multiply without affecting arithmetic flags.
PDEP	Parallel deposit of bits using a mask.
PEXT	Parallel extraction of bits using a mask.
RORX	Rotate right without affecting arithmetic flags.
SARX	Shift arithmetic right.
SHLX	Shift logic left.
SHRX	Shift logic right.
TZCNT	Count the number of trailing zero bits.

5.1.16.1 Detection of VEX-Encoded GPR Instructions, LZCNT, TZCNT, and PREFETCHW

VEX-encoded general-purpose instructions do not operate on any vector registers.

There are separate feature flags for the following subsets of instructions that operate on general purpose registers, and the detection requirements for hardware support are:

CPUID.07H.00H:EBX.BMI1[3]: if 1 indicates the processor supports the first group of advanced bit manipulation extensions (ANDN, BEXTR, BLSI, BLSMSK, BLSR, TZCNT);

CPUID.07H.00H:EBX.BMI2[8]: if 1 indicates the processor supports the second group of advanced bit manipulation extensions (BZHI, MULX, PDEP, PEXT, RORX, SARX, SHLX, SHRX);

CPUID.80000001H:ECX.LZCNT[5]: if 1 indicates the processor supports the LZCNT instruction.

CPUID.80000001H:ECX.PREFETCHW[8]: if 1 indicates the processor supports the PREFETCHW instruction.
 CPUID.07H.00H:ECX.PREFETCHWT1[0]: if 1 indicates the processor supports the PREFETCHWT1 instruction.

5.2 X87 FPU INSTRUCTIONS

The x87 FPU instructions are executed by the processor's x87 FPU. These instructions operate on floating-point, integer, and binary-coded decimal (BCD) operands. For more detail on x87 FPU instructions, see Chapter 8, "Programming with the x87 FPU."

These instructions are divided into the following subgroups: data transfer, load constants, and FPU control instructions. The sections that follow introduce each subgroup.

5.2.1 X87 FPU Data Transfer Instructions

The data transfer instructions move floating-point, integer, and BCD values between memory and the x87 FPU registers. They also perform conditional move operations on floating-point operands.

FLD	Load floating-point value.
FST	Store floating-point value.
FSTP	Store floating-point value and pop.
FILD	Load integer.
FIST	Store integer.
FISTP ¹	Store integer and pop.
FBLD	Load BCD.
FBSTP	Store BCD and pop.
FXCH	Exchange registers.
FCMOVE	Floating-point conditional move if equal.
FCMOVNE	Floating-point conditional move if not equal.
FCMOVB	Floating-point conditional move if below.
FCMOVBE	Floating-point conditional move if below or equal.
FCMOVNB	Floating-point conditional move if not below.
FCMOVNBE	Floating-point conditional move if not below or equal.
FCMOVU	Floating-point conditional move if unordered.
FCMOVNU	Floating-point conditional move if not unordered.

5.2.2 X87 FPU Basic Arithmetic Instructions

The basic arithmetic instructions perform basic arithmetic operations on floating-point and integer operands.

FADD	Add floating-point.
FADDP	Add floating-point and pop.
FIADD	Add integer.
FSUB	Subtract floating-point.
FSUBP	Subtract floating-point and pop.
FISUB	Subtract integer.
FSUBR	Subtract floating-point reverse.
FSUBRP	Subtract floating-point reverse and pop.
FISUBR	Subtract integer reverse.

1. SSE3 provides an instruction FISTTP for integer conversion.

FMUL	Multiply floating-point.
FMULP	Multiply floating-point and pop.
FIMUL	Multiply integer.
FDIV	Divide floating-point.
FDIVP	Divide floating-point and pop.
FIDIV	Divide integer.
FDIVR	Divide floating-point reverse.
FDIVRP	Divide floating-point reverse and pop.
FIDIVR	Divide integer reverse.
FPREM	Partial remainder.
FPREM1	IEEE partial remainder.
FABS	Absolute value.
FCHS	Change sign.
FRNDINT	Round to integer.
FSCALE	Scale by power of two.
FSQRT	Square root.
FXTRACT	Extract exponent and significand.

5.2.3 X87 FPU Comparison Instructions

The compare instructions examine or compare floating-point or integer operands.

FCOM	Compare floating-point.
FCOMP	Compare floating-point and pop.
FCOMPP	Compare floating-point and pop twice.
FUCOM	Unordered compare floating-point.
FUCOMP	Unordered compare floating-point and pop.
FUCOMPP	Unordered compare floating-point and pop twice.
FICOM	Compare integer.
FICOMP	Compare integer and pop.
FCOMI	Compare floating-point and set EFLAGS.
FUCOMI	Unordered compare floating-point and set EFLAGS.
FCOMIP	Compare floating-point, set EFLAGS, and pop.
FUCOMIP	Unordered compare floating-point, set EFLAGS, and pop.
FTST	Test floating-point (compare with 0.0).
FXAM	Examine floating-point.

5.2.4 X87 FPU Transcendental Instructions

The transcendental instructions perform basic trigonometric and logarithmic operations on floating-point operands.

FSIN	Sine.
FCOS	Cosine.
FSINCOS	Sine and cosine.
FPTAN	Partial tangent.
FPATAN	Partial arctangent.
F2XM1	$2^x - 1$.
FYL2X	$y * \log_2 x$.
FYL2XP1	$y * \log_2(x + 1)$.

5.2.5 X87 FPU Load Constants Instructions

The load constants instructions load common constants, such as π , into the x87 floating-point registers.

FLD1	Load +1.0.
FLDZ	Load +0.0.
FLDPI	Load π .
FLDL2E	Load $\log_2 e$.
FLDLN2	Load $\log_e 2$.
FLDL2T	Load $\log_2 10$.
FLDLG2	Load $\log_{10} 2$.

5.2.6 X87 FPU Control Instructions

The x87 FPU control instructions operate on the x87 FPU register stack and save and restore the x87 FPU state.

FINCSTP	Increment FPU register stack pointer.
FDECSTP	Decrement FPU register stack pointer.
FFREE	Free floating-point register.
FINIT	Initialize FPU after checking error conditions.
FNINIT	Initialize FPU without checking error conditions.
FCLEX	Clear floating-point exception flags after checking for error conditions.
FNCLEX	Clear floating-point exception flags without checking for error conditions.
FSTCW	Store FPU control word after checking error conditions.
FNSTCW	Store FPU control word without checking error conditions.
FLDCW	Load FPU control word.
FSTENV	Store FPU environment after checking error conditions.
FNSTENV	Store FPU environment without checking error conditions.
FLDENV	Load FPU environment.
FSAVE	Save FPU state after checking error conditions.
FNSAVE	Save FPU state without checking error conditions.
FRSTOR	Restore FPU state.
FSTSW	Store FPU status word after checking error conditions.
FNSTSW	Store FPU status word without checking error conditions.
WAIT/FWAIT	Wait for FPU.
FNOP	FPU no operation.

5.3 X87 FPU AND SIMD STATE MANAGEMENT INSTRUCTIONS

Two state management instructions were introduced into the IA-32 architecture with the Pentium II processor family:

FXSAVE	Save x87 FPU and SIMD state.
FXRSTOR	Restore x87 FPU and SIMD state.

Initially, these instructions operated only on the x87 FPU (and MMX) registers to perform a fast save and restore, respectively, of the x87 FPU and MMX state. With the introduction of SSE extensions in the Pentium III processor family, these instructions were expanded to also save and restore the state of the XMM and MXCSR registers. Intel 64 architecture also supports these instructions.

See Section 10.5, “FXSAVE and FXRSTOR Instructions,” for more detail.

5.4 MMX INSTRUCTIONS

Four extensions have been introduced into the IA-32 architecture to permit IA-32 processors to perform single-instruction multiple-data (SIMD) operations. These extensions include the MMX technology, SSE extensions, SSE2 extensions, and SSE3 extensions. For a discussion that puts SIMD instructions in their historical context, see Section 2.2.7, “SIMD Instructions.”

MMX instructions operate on packed byte, word, doubleword, or quadword integer operands contained in memory, in MMX registers, and/or in general-purpose registers. For more detail on these instructions, see Chapter 9, “Programming with Intel® MMX™ Technology.”

MMX instructions can only be executed on Intel 64 and IA-32 processors that support the MMX technology. Support for these instructions can be detected with the CPUID instruction. See the description of the CPUID instruction in Chapter 3, “Instruction Set Reference, A-L,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.

MMX instructions are divided into the following subgroups: data transfer, conversion, packed arithmetic, comparison, logical, shift and rotate, and state management instructions. The sections that follow introduce each subgroup.

5.4.1 MMX Data Transfer Instructions

The data transfer instructions move doubleword and quadword operands between MMX registers and between MMX registers and memory.

MOVD	Move doubleword.
MOVQ	Move quadword.

5.4.2 MMX Conversion Instructions

The conversion instructions pack and unpack bytes, words, and doublewords

PACKSSWB	Pack words into bytes with signed saturation.
PACKSSDW	Pack doublewords into words with signed saturation.
PACKUSWB	Pack words into bytes with unsigned saturation.
PUNPCKHBW	Unpack high-order bytes.
PUNPCKHWD	Unpack high-order words.
PUNPCKHDQ	Unpack high-order doublewords.
PUNPCKLBW	Unpack low-order bytes.
PUNPCKLWD	Unpack low-order words.
PUNPCKLDQ	Unpack low-order doublewords.

5.4.3 MMX Packed Arithmetic Instructions

The packed arithmetic instructions perform packed integer arithmetic on packed byte, word, and doubleword integers.

PADDB	Add packed byte integers.
PADDW	Add packed word integers.
PADDQ	Add packed doubleword integers.
PADDSB	Add packed signed byte integers with signed saturation.
PADDSD	Add packed signed word integers with signed saturation.
PADDUSB	Add packed unsigned byte integers with unsigned saturation.
PADDUSW	Add packed unsigned word integers with unsigned saturation.
PSUBB	Subtract packed byte integers.
PSUBW	Subtract packed word integers.

PSUBD	Subtract packed doubleword integers.
PSUBSB	Subtract packed signed byte integers with signed saturation.
PSUBSW	Subtract packed signed word integers with signed saturation.
PSUBUSB	Subtract packed unsigned byte integers with unsigned saturation.
PSUBUSW	Subtract packed unsigned word integers with unsigned saturation.
PMULHW	Multiply packed signed word integers and store high result.
PMULLW	Multiply packed signed word integers and store low result.
PMADDWD	Multiply and add packed word integers.

5.4.4 MMX Comparison Instructions

The compare instructions compare packed bytes, words, or doublewords.

PCMPEQB	Compare packed bytes for equal.
PCMPEQW	Compare packed words for equal.
PCMPEQD	Compare packed doublewords for equal.
PCMPGTB	Compare packed signed byte integers for greater than.
PCMPGTW	Compare packed signed word integers for greater than.
PCMPGTD	Compare packed signed doubleword integers for greater than.

5.4.5 MMX Logical Instructions

The logical instructions perform AND, AND NOT, OR, and XOR operations on quadword operands.

PAND	Bitwise logical AND.
PANDN	Bitwise logical AND NOT.
POR	Bitwise logical OR.
PXOR	Bitwise logical exclusive OR.

5.4.6 MMX Shift and Rotate Instructions

The shift and rotate instructions shift and rotate packed bytes, words, or doublewords, or quadwords in 64-bit operands.

PSLLW	Shift packed words left logical.
PSLLD	Shift packed doublewords left logical.
PSLLQ	Shift packed quadword left logical.
PSRLW	Shift packed words right logical.
PSRLD	Shift packed doublewords right logical.
PSRLQ	Shift packed quadword right logical.
PSRAW	Shift packed words right arithmetic.
PSRAD	Shift packed doublewords right arithmetic.

5.4.7 MMX State Management Instructions

The EMMS instruction clears the MMX state from the MMX registers.

EMMS	Empty MMX state.

5.5 INTEL® SSE INSTRUCTIONS

Intel SSE instructions represent an extension of the SIMD execution model introduced with the MMX technology. For more detail on these instructions, see Chapter 10, “Programming with Intel® Streaming SIMD Extensions (Intel® SSE).”

Intel SSE instructions can only be executed on Intel 64 and IA-32 processors that support Intel SSE extensions. Support for these instructions can be detected with the CPUID instruction. See the description of the CPUID instruction in Chapter 3, “Instruction Set Reference, A-L,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.

Intel SSE instructions are divided into four subgroups (note that the first subgroup has subordinate subgroups of its own):

- SIMD single precision floating-point instructions that operate on the XMM registers.
- MXCSR state management instructions.
- 64-bit SIMD integer instructions that operate on the MMX registers.
- Cacheability control, prefetch, and instruction ordering instructions.

The following sections provide an overview of these groups.

5.5.1 Intel® SSE SIMD Single Precision Floating-Point Instructions

These instructions operate on packed and scalar single precision floating-point values located in XMM registers and/or memory. This subgroup is further divided into the following subordinate subgroups: data transfer, packed arithmetic, comparison, logical, shuffle and unpack, and conversion instructions.

5.5.1.1 Intel® SSE Data Transfer Instructions

Intel SSE data transfer instructions move packed and scalar single precision floating-point operands between XMM registers and between XMM registers and memory.

MOVAPS	Move four aligned packed single precision floating-point values between XMM registers or between an XMM register and memory.
MOVUPS	Move four unaligned packed single precision floating-point values between XMM registers or between an XMM register and memory.
MOVHPS	Move two packed single precision floating-point values to and from the high quadword of an XMM register and memory.
MOVHLPS	Move two packed single precision floating-point values from the high quadword of an XMM register to the low quadword of another XMM register.
MOVLPS	Move two packed single precision floating-point values to and from the low quadword of an XMM register and memory.
MOVLHPS	Move two packed single precision floating-point values from the low quadword of an XMM register to the high quadword of another XMM register.
MOVMSKPS	Extract sign mask from four packed single precision floating-point values.
MOVSS	Move scalar single precision floating-point value between XMM registers or between an XMM register and memory.

5.5.1.2 Intel® SSE Packed Arithmetic Instructions

Intel SSE packed arithmetic instructions perform packed and scalar arithmetic operations on packed and scalar single precision floating-point operands.

ADDPS	Add packed single precision floating-point values.
ADDSS	Add scalar single precision floating-point values.
SUBPS	Subtract packed single precision floating-point values.
SUBSS	Subtract scalar single precision floating-point values.

MULPS	Multiply packed single precision floating-point values.
MULSS	Multiply scalar single precision floating-point values.
DIVPS	Divide packed single precision floating-point values.
DIVSS	Divide scalar single precision floating-point values.
RCPPS	Compute reciprocals of packed single precision floating-point values.
RCPSS	Compute reciprocal of scalar single precision floating-point values.
SQRTPS	Compute square roots of packed single precision floating-point values.
SQRTSS	Compute square root of scalar single precision floating-point values.
RSQRTPS	Compute reciprocals of square roots of packed single precision floating-point values.
RSQRTSS	Compute reciprocal of square root of scalar single precision floating-point values.
MAXPS	Return maximum packed single precision floating-point values.
MAXSS	Return maximum scalar single precision floating-point values.
MINPS	Return minimum packed single precision floating-point values.
MINSS	Return minimum scalar single precision floating-point values.

5.5.1.3 Intel® SSE Comparison Instructions

Intel SSE compare instructions compare packed and scalar single precision floating-point operands.

CMPPS	Compare packed single precision floating-point values.
CMPSS	Compare scalar single precision floating-point values.
COMISS	Perform ordered comparison of scalar single precision floating-point values and set flags in EFLAGS register.
UCOMISS	Perform unordered comparison of scalar single precision floating-point values and set flags in EFLAGS register.

5.5.1.4 Intel® SSE Logical Instructions

Intel SSE logical instructions perform bitwise AND, AND NOT, OR, and XOR operations on packed single precision floating-point operands.

ANDPS	Perform bitwise logical AND of packed single precision floating-point values.
ANDNPS	Perform bitwise logical AND NOT of packed single precision floating-point values.
ORPS	Perform bitwise logical OR of packed single precision floating-point values.
XORPS	Perform bitwise logical XOR of packed single precision floating-point values.

5.5.1.5 Intel® SSE Shuffle and Unpack Instructions

Intel SSE shuffle and unpack instructions shuffle or interleave single precision floating-point values in packed single precision floating-point operands.

SHUFPS	Shuffles values in packed single precision floating-point operands.
UNPCKHPS	Unpacks and interleaves the two high-order values from two single precision floating-point operands.
UNPCKLPS	Unpacks and interleaves the two low-order values from two single precision floating-point operands.

5.5.1.6 Intel® SSE Conversion Instructions

Intel SSE conversion instructions convert packed and individual doubleword integers into packed and scalar single precision floating-point values and vice versa.

CVTPI2PS	Convert packed doubleword integers to packed single precision floating-point values.
CVTSI2SS	Convert signed integer to scalar single precision floating-point value.

CVTSP2PI	Convert packed single precision floating-point values to packed doubleword integers.
CVTTP2PI	Convert with truncation packed single precision floating-point values to packed doubleword integers.
CVTSS2SI	Convert a scalar single precision floating-point value to a signed integer.
CVTSS2SI	Convert with truncation a scalar single precision floating-point value to a scalar signed integer.

5.5.2 Intel® SSE MXCSR State Management Instructions

MXCSR state management instructions allow saving and restoring the state of the MXCSR control and status register.

LDMXCSR	Load MXCSR register.
STMXCSR	Save MXCSR register state.

5.5.3 Intel® SSE 64-Bit SIMD Integer Instructions

These Intel SSE 64-bit SIMD integer instructions perform additional operations on packed bytes, words, or doublewords contained in MMX registers. They represent enhancements to the MMX instruction set described in Section 5.4, “MMX Instructions.”

PAVGB	Compute average of packed unsigned byte integers.
PAVGW	Compute average of packed unsigned word integers.
PEXTRW	Extract word.
PINSRW	Insert word.
PMAXUB	Maximum of packed unsigned byte integers.
PMAXSW	Maximum of packed signed word integers.
PMINUB	Minimum of packed unsigned byte integers.
PMINSW	Minimum of packed signed word integers.
PMOVMASKB	Move byte mask.
PMULHUW	Multiply packed unsigned integers and store high result.
PSADBW	Compute sum of absolute differences.
PSHUFW	Shuffle packed integer word in MMX register.

5.5.4 Intel® SSE Cacheability Control, Prefetch, and Instruction Ordering Instructions

The cacheability control instructions provide control over the caching of non-temporal data when storing data from the MMX and XMM registers to memory. The `PREFETCHh` allows data to be prefetched to a selected cache level. The `SFENCE` instruction controls instruction ordering on store operations.

MASKMOVQ	Non-temporal store of selected bytes from an MMX register into memory.
MOVNTQ	Non-temporal store of quadword from an MMX register into memory.
MOVNTPS	Non-temporal store of four packed single precision floating-point values from an XMM register into memory.
PREFETCHh	Load 32 or more of bytes from memory to a selected level of the processor’s cache hierarchy.
SFENCE	Serializes store operations.

5.6 INTEL® SSE2 INSTRUCTIONS

Intel SSE2 extensions represent an extension of the SIMD execution model introduced with MMX technology and the Intel SSE extensions. Intel SSE2 instructions operate on packed double precision floating-point operands and

on packed byte, word, doubleword, and quadword operands located in the XMM registers. For more detail on these instructions, see Chapter 11, “Programming with Intel® Streaming SIMD Extensions 2 (Intel® SSE2).”

Intel SSE2 instructions can only be executed on Intel 64 and IA-32 processors that support the Intel SSE2 extensions. Support for these instructions can be detected with the CPUID instruction. See the description of the CPUID instruction in Chapter 3, “Instruction Set Reference, A-L,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.

These instructions are divided into four subgroups (note that the first subgroup is further divided into subordinate subgroups):

- Packed and scalar double precision floating-point instructions.
- Packed single precision floating-point conversion instructions.
- 128-bit SIMD integer instructions.
- Cacheability-control and instruction ordering instructions.

The following sections give an overview of each subgroup.

5.6.1 Intel® SSE2 Packed and Scalar Double Precision Floating-Point Instructions

Intel SSE2 packed and scalar double precision floating-point instructions are divided into the following subordinate subgroups: data movement, arithmetic, comparison, conversion, logical, and shuffle operations on double precision floating-point operands. These are introduced in the sections that follow.

5.6.1.1 Intel® SSE2 Data Movement Instructions

Intel SSE2 data movement instructions move double precision floating-point data between XMM registers and between XMM registers and memory.

MOVAPD	Move two aligned packed double precision floating-point values between XMM registers or between an XMM register and memory.
MOVUPD	Move two unaligned packed double precision floating-point values between XMM registers or between an XMM register and memory.
MOVHPD	Move high packed double precision floating-point value to and from the high quadword of an XMM register and memory.
MOVLPD	Move low packed single precision floating-point value to and from the low quadword of an XMM register and memory.
MOVMSKPD	Extract sign mask from two packed double precision floating-point values.
MOVSD	Move scalar double precision floating-point value between XMM registers or between an XMM register and memory.

5.6.1.2 Intel® SSE2 Packed Arithmetic Instructions

The arithmetic instructions perform addition, subtraction, multiply, divide, square root, and maximum/minimum operations on packed and scalar double precision floating-point operands.

ADDPD	Add packed double precision floating-point values.
ADDSD	Add scalar double precision floating-point values.
SUBPD	Subtract packed double precision floating-point values.
SUBSD	Subtract scalar double precision floating-point values.
MULPD	Multiply packed double precision floating-point values.
MULSD	Multiply scalar double precision floating-point values.
DIVPD	Divide packed double precision floating-point values.
DIVSD	Divide scalar double precision floating-point values.
SQRTPD	Compute packed square roots of packed double precision floating-point values.
SQRTSD	Compute scalar square root of scalar double precision floating-point values.

MAXPD	Return maximum packed double precision floating-point values.
MAXSD	Return maximum scalar double precision floating-point values.
MINPD	Return minimum packed double precision floating-point values.
MINSD	Return minimum scalar double precision floating-point values.

5.6.1.3 Intel® SSE2 Logical Instructions

Intel SSE2 logical instructions perform AND, AND NOT, OR, and XOR operations on packed double precision floating-point values.

ANDPD	Perform bitwise logical AND of packed double precision floating-point values.
ANDNPD	Perform bitwise logical AND NOT of packed double precision floating-point values.
ORPD	Perform bitwise logical OR of packed double precision floating-point values.
XORPD	Perform bitwise logical XOR of packed double precision floating-point values.

5.6.1.4 Intel® SSE2 Compare Instructions

Intel SSE2 compare instructions compare packed and scalar double precision floating-point values and return the results of the comparison either to the destination operand or to the EFLAGS register.

CMPPD	Compare packed double precision floating-point values.
CMPSD	Compare scalar double precision floating-point values.
COMISD	Perform ordered comparison of scalar double precision floating-point values and set flags in EFLAGS register.
UCOMISD	Perform unordered comparison of scalar double precision floating-point values and set flags in EFLAGS register.

5.6.1.5 Intel® SSE2 Shuffle and Unpack Instructions

Intel SSE2 shuffle and unpack instructions shuffle or interleave double precision floating-point values in packed double precision floating-point operands.

SHUFPD	Shuffles values in packed double precision floating-point operands.
UNPCKHPD	Unpacks and interleaves the high values from two packed double precision floating-point operands.
UNPCKLPD	Unpacks and interleaves the low values from two packed double precision floating-point operands.

5.6.1.6 Intel® SSE2 Conversion Instructions

Intel SSE2 conversion instructions convert packed and individual doubleword integers into packed and scalar double precision floating-point values and vice versa. They also convert between packed and scalar single precision and double precision floating-point values.

CVTPD2PI	Convert packed double precision floating-point values to packed doubleword integers.
CVTTPD2PI	Convert with truncation packed double precision floating-point values to packed doubleword integers.
CVTPI2PD	Convert packed doubleword integers to packed double precision floating-point values.
CVTPD2DQ	Convert packed double precision floating-point values to packed doubleword integers.
CVTTPD2DQ	Convert with truncation packed double precision floating-point values to packed doubleword integers.
CVTDQ2PD	Convert packed doubleword integers to packed double precision floating-point values.
CVTPS2PD	Convert packed single precision floating-point values to packed double precision floating-point values.

CVTPD2PS	Convert packed double precision floating-point values to packed single precision floating-point values.
CVTSS2SD	Convert scalar single precision floating-point values to scalar double precision floating-point values.
CVTSD2SS	Convert scalar double precision floating-point values to scalar single precision floating-point values.
CVTSD2SI	Convert scalar double precision floating-point values to a signed integer.
CVTTSD2SI	Convert with truncation scalar double precision floating-point values to a scalar signed integer.
CVTSI2SD	Convert signed integer to scalar double precision floating-point value.

5.6.2 Intel® SSE2 Packed Single Precision Floating-Point Instructions

Intel SSE2 packed single precision floating-point instructions perform conversion operations on single precision floating-point and integer operands. These instructions represent enhancements to the Intel SSE single precision floating-point instructions.

CVTDQ2PS	Convert packed doubleword integers to packed single precision floating-point values.
CVTPS2DQ	Convert packed single precision floating-point values to packed doubleword integers.
CVTTPS2DQ	Convert with truncation packed single precision floating-point values to packed doubleword integers.

5.6.3 Intel® SSE2 128-Bit SIMD Integer Instructions

Intel SSE2 SIMD integer instructions perform additional operations on packed words, doublewords, and quadwords contained in XMM and MMX registers.

MOVDQA	Move aligned double quadword.
MOVDQU	Move unaligned double quadword.
MOVQ2DQ	Move quadword integer from MMX to XMM registers.
MOVDQ2Q	Move quadword integer from XMM to MMX registers.
PMULUDQ	Multiply packed unsigned doubleword integers.
PADDQ	Add packed quadword integers.
PSUBQ	Subtract packed quadword integers.
PSHUFLW	Shuffle packed low words.
PSHUFHW	Shuffle packed high words.
PSHUFDB	Shuffle packed doublewords.
PSLLDQ	Shift double quadword left logical.
PSRLDQ	Shift double quadword right logical.
PUNPCKHQDQ	Unpack high quadwords.
PUNPCKLQDQ	Unpack low quadwords.

5.6.4 Intel® SSE2 Cacheability Control and Ordering Instructions

Intel SSE2 cacheability control instructions provide additional operations for caching of non-temporal data when storing data from XMM registers to memory. LFENCE and MFENCE provide additional control of instruction ordering on store operations.

CLFLUSH	See Section 5.1.13.
LFENCE	Serializes load operations.
MFENCE	Serializes load and store operations.
PAUSE	Improves the performance of “spin-wait loops”.

MASKMOVDQU	Non-temporal store of selected bytes from an XMM register into memory.
MOVNTPD	Non-temporal store of two packed double precision floating-point values from an XMM register into memory.
MOVNTDQ	Non-temporal store of double quadword from an XMM register into memory.
MOVNTI	Non-temporal store of a doubleword from a general-purpose register into memory.

5.7 INTEL® SSE3 INSTRUCTIONS

The Intel SSE3 extensions offers 13 instructions that accelerate performance of Streaming SIMD Extensions technology, Streaming SIMD Extensions 2 technology, and x87-FP math capabilities. These instructions can be grouped into the following categories:

- One x87 FPU instruction used in integer conversion.
- One SIMD integer instruction that addresses unaligned data loads.
- Two SIMD floating-point packed ADD/SUB instructions.
- Four SIMD floating-point horizontal ADD/SUB instructions.
- Three SIMD floating-point LOAD/MOVE/DUPLICATE instructions.
- Two thread synchronization instructions.

Intel SSE3 instructions can only be executed on Intel 64 and IA-32 processors that support Intel SSE3 extensions. Support for these instructions can be detected with the CPUID instruction. See the description of the CPUID instruction in Chapter 3, “Instruction Set Reference, A-L,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.

The sections that follow describe each subgroup.

5.7.1 Intel® SSE3 x87-FP Integer Conversion Instruction

FISTTP	Behaves like the FISTP instruction but uses truncation, irrespective of the rounding mode specified in the floating-point control word (FCW).

5.7.2 Intel® SSE3 Specialized 128-Bit Unaligned Data Load Instruction

LDDQU	Special 128-bit unaligned load designed to avoid cache line splits.

5.7.3 Intel® SSE3 SIMD Floating-Point Packed ADD/SUB Instructions

ADDSUBPS	Performs single precision addition on the second and fourth pairs of 32-bit data elements within the operands; single precision subtraction on the first and third pairs.
ADDSUBPD	Performs double precision addition on the second pair of quadwords, and double precision subtraction on the first pair.

5.7.4 Intel® SSE3 SIMD Floating-Point Horizontal ADD/SUB Instructions

HADDPS	Performs a single precision addition on contiguous data elements. The first data element of the result is obtained by adding the first and second elements of the first operand; the second element by adding the third and fourth elements of the first operand; the third by adding the first and second elements of the second operand; and the fourth by adding the third and fourth elements of the second operand.
HSUBPS	Performs a single precision subtraction on contiguous data elements. The first data element of the result is obtained by subtracting the second element of the first operand from the first element of the first operand; the second element by subtracting the fourth element of the first operand from the third element of the first operand; the third by

subtracting the second element of the second operand from the first element of the second operand; and the fourth by subtracting the fourth element of the second operand from the third element of the second operand.

HADDPD	Performs a double precision addition on contiguous data elements. The first data element of the result is obtained by adding the first and second elements of the first operand; the second element by adding the first and second elements of the second operand.
HSUBPD	Performs a double precision subtraction on contiguous data elements. The first data element of the result is obtained by subtracting the second element of the first operand from the first element of the first operand; the second element by subtracting the second element of the second operand from the first element of the second operand.

5.7.5 Intel® SSE3 SIMD Floating-Point LOAD/MOVE/DUPLICATE Instructions

MOVSHDUP	Loads/moves 128 bits; duplicating the second and fourth 32-bit data elements.
MOVSLDUP	Loads/moves 128 bits; duplicating the first and third 32-bit data elements.
MOVDDUP	Loads/moves 64 bits (bits[63:0] if the source is a register) and returns the same 64 bits in both the lower and upper halves of the 128-bit result register; duplicates the 64 bits from the source.

5.7.6 Intel® SSE3 Agent Synchronization Instructions

MONITOR	Sets up an address range used to monitor write-back stores.
MWAIT	Enables a logical processor to enter into an optimized state while waiting for a write-back store to the address range set up by the MONITOR instruction.

5.8 SUPPLEMENTAL STREAMING SIMD EXTENSIONS 3 (SSSE3) INSTRUCTIONS

SSSE3 provide 32 instructions (represented by 14 mnemonics) to accelerate computations on packed integers. These include:

- Twelve instructions that perform horizontal addition or subtraction operations.
- Six instructions that evaluate absolute values.
- Two instructions that perform multiply and add operations and speed up the evaluation of dot products.
- Two instructions that accelerate packed-integer multiply operations and produce integer values with scaling.
- Two instructions that perform a byte-wise, in-place shuffle according to the second shuffle control operand.
- Six instructions that negate packed integers in the destination operand if the signs of the corresponding element in the source operand is less than zero.
- Two instructions that align data from the composite of two operands.

SSSE3 instructions can only be executed on Intel 64 and IA-32 processors that support SSSE3 extensions. Support for these instructions can be detected with the CPUID instruction. See the description of the CPUID instruction in Chapter 3, “Instruction Set Reference, A-L,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.

The sections that follow describe each subgroup.

5.8.1 Horizontal Addition/Subtraction

PHADDW	Adds two adjacent, signed 16-bit integers horizontally from the source and destination operands and packs the signed 16-bit results to the destination operand.
PHADDSW	Adds two adjacent, signed 16-bit integers horizontally from the source and destination operands and packs the signed, saturated 16-bit results to the destination operand.

PHADD	Adds two adjacent, signed 32-bit integers horizontally from the source and destination operands and packs the signed 32-bit results to the destination operand.
PHSUBW	Performs horizontal subtraction on each adjacent pair of 16-bit signed integers by subtracting the most significant word from the least significant word of each pair in the source and destination operands. The signed 16-bit results are packed and written to the destination operand.
PHSUBSW	Performs horizontal subtraction on each adjacent pair of 16-bit signed integers by subtracting the most significant word from the least significant word of each pair in the source and destination operands. The signed, saturated 16-bit results are packed and written to the destination operand.
PHSUBD	Performs horizontal subtraction on each adjacent pair of 32-bit signed integers by subtracting the most significant doubleword from the least significant double word of each pair in the source and destination operands. The signed 32-bit results are packed and written to the destination operand.

5.8.2 Packed Absolute Values

PABSB	Computes the absolute value of each signed byte data element.
PABSW	Computes the absolute value of each signed 16-bit data element.
PABSD	Computes the absolute value of each signed 32-bit data element.

5.8.3 Multiply and Add Packed Signed and Unsigned Bytes

PMADDUSW	Multiplies each unsigned byte value with the corresponding signed byte value to produce an intermediate, 16-bit signed integer. Each adjacent pair of 16-bit signed values are added horizontally. The signed, saturated 16-bit results are packed to the destination operand.

5.8.4 Packed Multiply High with Round and Scale

PMULHRW	Multiplies vertically each signed 16-bit integer from the destination operand with the corresponding signed 16-bit integer of the source operand, producing intermediate, signed 32-bit integers. Each intermediate 32-bit integer is truncated to the 18 most significant bits. Rounding is always performed by adding 1 to the least significant bit of the 18-bit intermediate result. The final result is obtained by selecting the 16 bits immediately to the right of the most significant bit of each 18-bit intermediate result and packed to the destination operand.

5.8.5 Packed Shuffle Bytes

PSHUF	Permutes each byte in place, according to a shuffle control mask. The least significant three or four bits of each shuffle control byte of the control mask form the shuffle index. The shuffle mask is unaffected. If the most significant bit (bit 7) of a shuffle control byte is set, the constant zero is written in the result byte.

5.8.6 Packed Sign

PSIGNB/W/D	Negates each signed integer element of the destination operand if the sign of the corresponding data element in the source operand is less than zero.

5.8.7 Packed Align Right

PALIGNR

Source operand is appended after the destination operand forming an intermediate value of twice the width of an operand. The result is extracted from the intermediate value into the destination operand by selecting the 128-bit or 64-bit value that are right-aligned to the byte offset specified by the immediate value.

5.9 INTEL® SSE4 INSTRUCTIONS

Intel Streaming SIMD Extensions 4 (Intel SSE4) introduces 54 new instructions. 47 of the Intel SSE4 instructions are referred to as Intel SSE4.1 in this document, and 7 new Intel SSE4 instructions are referred to as Intel SSE4.2.

Intel SSE4.1 is targeted to improve the performance of media, imaging, and 3D workloads. Intel SSE4.1 adds instructions that improve compiler vectorization and significantly increase support for packed dword computation. The technology also provides a hint that can improve memory throughput when reading from uncacheable WC memory type.

The 47 Intel SSE4.1 instructions include:

- Two instructions perform packed dword multiplies.
- Two instructions perform floating-point dot products with input/output selects.
- One instruction performs a load with a streaming hint.
- Six instructions simplify packed blending.
- Eight instructions expand support for packed integer MIN/MAX.
- Four instructions support floating-point round with selectable rounding mode and precision exception override.
- Seven instructions improve data insertion and extractions from XMM registers
- Twelve instructions improve packed integer format conversions (sign and zero extensions).
- One instruction improves SAD (sum absolute difference) generation for small block sizes.
- One instruction aids horizontal searching operations.
- One instruction improves masked comparisons.
- One instruction adds qword packed equality comparisons.
- One instruction adds dword packing with unsigned saturation.

The Intel SSE4.2 instructions operating on XMM registers include:

- String and text processing that can take advantage of single-instruction multiple-data programming techniques.
- A SIMD integer instruction that enhances the capability of the 128-bit integer SIMD capability in SSE4.1.

5.10 INTEL® SSE4.1 INSTRUCTIONS

Intel SSE4.1 instructions can use an XMM register as a source or destination. Programming Intel SSE4.1 is similar to programming 128-bit Integer SIMD and floating-point SIMD instructions in Intel SSE/SSE2/SSE3/SSSE3. Intel SSE4.1 does not provide any 64-bit integer SIMD instructions operating on MMX registers. The sections that follow describe each subgroup.

5.10.1 Dword Multiply Instructions

PMULLD

Returns four lower 32-bits of the 64-bit results of signed 32-bit integer multiplies.

PMULDQ

Returns two 64-bit signed result of signed 32-bit integer multiplies.

5.10.2 Floating-Point Dot Product Instructions

DPPD	Perform double precision dot product for up to 2 elements and broadcast.
DPPS	Perform single precision dot products for up to 4 elements and broadcast.

5.10.3 Streaming Load Hint Instruction

MOVNTDQA	Provides a non-temporal hint that can cause adjacent 16-byte items within an aligned 64-byte region (a streaming line) to be fetched and held in a small set of temporary buffers ("streaming load buffers"). Subsequent streaming loads to other aligned 16-byte items in the same streaming line may be supplied from the streaming load buffer and can improve throughput.

5.10.4 Packed Blending Instructions

BLENDPD	Conditionally copies specified double precision floating-point data elements in the source operand to the corresponding data elements in the destination, using an immediate byte control.
BLENDPS	Conditionally copies specified single precision floating-point data elements in the source operand to the corresponding data elements in the destination, using an immediate byte control.
BLENDVPD	Conditionally copies specified double precision floating-point data elements in the source operand to the corresponding data elements in the destination, using an implied mask.
BLENDVPS	Conditionally copies specified single precision floating-point data elements in the source operand to the corresponding data elements in the destination, using an implied mask.
PBLENDVB	Conditionally copies specified byte elements in the source operand to the corresponding elements in the destination, using an implied mask.
PBLENDW	Conditionally copies specified word elements in the source operand to the corresponding elements in the destination, using an immediate byte control.

5.10.5 Packed Integer MIN/MAX Instructions

PMINUW	Compare packed unsigned word integers.
PMINUD	Compare packed unsigned dword integers.
PMINSB	Compare packed signed byte integers.
PMINSW	Compare packed signed dword integers.
PMAXUW	Compare packed unsigned word integers.
PMAXUD	Compare packed unsigned dword integers.
PMAXSB	Compare packed signed byte integers.
PMAXSD	Compare packed signed dword integers.

5.10.6 Floating-Point Round Instructions With Selectable Rounding Mode

ROUNDPS	Round packed single precision floating-point values into integer values and return rounded floating-point values.
ROUNDPD	Round packed double precision floating-point values into integer values and return rounded floating-point values.
ROUNDSS	Round the low packed single precision floating-point value into an integer value and return a rounded floating-point value.
ROUNDSD	Round the low packed double precision floating-point value into an integer value and return a rounded floating-point value.

5.10.7 Insertion and Extractions from XMM Registers

EXTRACTPS	Extracts a single precision floating-point value from a specified offset in an XMM register and stores the result to memory or a general-purpose register.
INSERTPS	Inserts a single precision floating-point value from either a 32-bit memory location or selected from a specified offset in an XMM register to a specified offset in the destination XMM register. In addition, INSERTPS allows zeroing out selected data elements in the destination, using a mask.
PINSRB	Insert a byte value from a register or memory into an XMM register.
PINSRD	Insert a dword value from 32-bit register or memory into an XMM register.
PINSRQ	Insert a qword value from 64-bit register or memory into an XMM register.
PEXTRB	Extract a byte from an XMM register and insert the value into a general-purpose register or memory.
PEXTRW	Extract a word from an XMM register and insert the value into a general-purpose register or memory.
PEXTRD	Extract a dword from an XMM register and insert the value into a general-purpose register or memory.
PEXTRQ	Extract a qword from an XMM register and insert the value into a general-purpose register or memory.

5.10.8 Packed Integer Format Conversions

PMOVSXBW	Sign extend the lower 8-bit integer of each packed word element into packed signed word integers.
PMOVZXBW	Zero extend the lower 8-bit integer of each packed word element into packed signed word integers.
PMOVSXBD	Sign extend the lower 8-bit integer of each packed dword element into packed signed dword integers.
PMOVZXBBD	Zero extend the lower 8-bit integer of each packed dword element into packed signed dword integers.
PMOVSXWD	Sign extend the lower 16-bit integer of each packed dword element into packed signed dword integers.
PMOVZXWD	Zero extend the lower 16-bit integer of each packed dword element into packed signed dword integers.
PMOVSXBQ	Sign extend the lower 8-bit integer of each packed qword element into packed signed qword integers.
PMOVZXBQ	Zero extend the lower 8-bit integer of each packed qword element into packed signed qword integers.
PMOVSXWQ	Sign extend the lower 16-bit integer of each packed qword element into packed signed qword integers.
PMOVZXWQ	Zero extend the lower 16-bit integer of each packed qword element into packed signed qword integers.
PMOVSXDQ	Sign extend the lower 32-bit integer of each packed qword element into packed signed qword integers.
PMOVZXDQ	Zero extend the lower 32-bit integer of each packed qword element into packed signed qword integers.

5.10.9 Improved Sums of Absolute Differences (SAD) for 4-Byte Blocks

MPSADBW	Performs eight 4-byte wide Sum of Absolute Differences operations to produce eight word integers.

5.10.10 Horizontal Search

PHMINPOSUW Finds the value and location of the minimum unsigned word from one of 8 horizontally packed unsigned words. The resulting value and location (offset within the source) are packed into the low dword of the destination XMM register.

5.10.11 Packed Test

PTEST Performs a logical AND between the destination with this mask and sets the ZF flag if the result is zero. The CF flag (zero for TEST) is set if the inverted mask AND'd with the destination is all zeroes.

5.10.12 Packed Qword Equality Comparisons

PCMPEQQ 128-bit packed qword equality test.

5.10.13 Dword Packing With Unsigned Saturation

PACKUSDW Packs dword to word with unsigned saturation.

5.11 INTEL® SSE4.2 INSTRUCTION SET

Five of the Intel SSE4.2 instructions operate on XMM register as a source or destination. These include four text/string processing instructions and one packed quadword compare SIMD instruction. Programming these five Intel SSE4.2 instructions is similar to programming 128-bit Integer SIMD in Intel SSE2/SSSE3. Intel SSE4.2 does not provide any 64-bit integer SIMD instructions.

CRC32 operates on general-purpose registers and is summarized in Section 5.1.6. The sections that follow summarize each subgroup.

5.11.1 String and Text Processing Instructions

PCMPESTRI	Packed compare explicit-length strings, return index in ECX/RCX.
PCMPESTRM	Packed compare explicit-length strings, return mask in XMM0.
PCMPISTRI	Packed compare implicit-length strings, return index in ECX/RCX.
PCMPISTRM	Packed compare implicit-length strings, return mask in XMM0.

5.11.2 Packed Comparison SIMD Integer Instruction

PCMPGTQ Performs logical compare of greater-than on packed integer quadwords.

5.12 INTEL® AES-NI AND PCLMULQDQ

Six Intel® AES-NI instructions operate on XMM registers to provide accelerated primitives for block encryption/decryption using Advanced Encryption Standard (FIPS-197). The PCLMULQDQ instruction performs carry-less multiplication for two binary numbers up to 64-bit wide.

AESDEC	Perform an AES decryption round using an 128-bit state and a round key.
AESDECLAST	Perform the last AES decryption round using an 128-bit state and a round key.
AESENC	Perform an AES encryption round using an 128-bit state and a round key.
AESENCLAST	Perform the last AES encryption round using an 128-bit state and a round key.
AESIMC	Perform an inverse mix column transformation primitive.

AESKEYGENASSIST	Assist the creation of round keys with a key expansion schedule.
PCLMULQDQ	Perform carryless multiplication of two 64-bit numbers.

5.13 INTEL® ADVANCED VECTOR EXTENSIONS (INTEL® AVX)

Intel® Advanced Vector Extensions (AVX) promote legacy 128-bit SIMD instruction sets that operate on the XMM register set to use a “vector extension” (VEX) prefix and operates on 256-bit vector registers (YMM). Almost all prior generations of 128-bit SIMD instructions that operate on XMM (but not on MMX registers) are promoted to support three-operand syntax with VEX-128 encoding.

VEX-prefix encoded Intel AVX instructions support 256-bit and 128-bit floating-point operations by extending the legacy 128-bit SIMD floating-point instructions to support three-operand syntax.

Additional functional enhancements are also provided with VEX-encoded Intel AVX instructions.

The list of Intel AVX instructions is included in the following tables:

- Table 14-2 lists 256-bit and 128-bit floating-point arithmetic instructions promoted from legacy 128-bit SIMD instruction sets.
- Table 14-3 lists 256-bit and 128-bit data movement and processing instructions promoted from legacy 128-bit SIMD instruction sets.
- Table 14-4 lists functional enhancements of 256-bit Intel AVX instructions not available from legacy 128-bit SIMD instruction sets.
- Table 14-5 lists 128-bit integer and floating-point instructions promoted from legacy 128-bit SIMD instruction sets.
- Table 14-6 lists functional enhancements of 128-bit Intel AVX instructions not available from legacy 128-bit SIMD instruction sets.
- Table 14-7 lists 128-bit data movement and processing instructions promoted from legacy instruction sets.

5.14 16-BIT FLOATING-POINT CONVERSION

Conversions between single precision floating-point (32-bit) and half precision floating-point (16-bit) data are provided by the VCVTPS2PH and VCVTPH2PS instructions, introduced beginning with the third generation of Intel Core processors based on Ivy Bridge microarchitecture:

VCVTPH2PS	Convert eight/four data elements containing 16-bit floating-point data into eight/four single precision floating-point data.
VCVTPS2PH	Convert eight/four data elements containing single precision floating-point data into eight/four 16-bit floating-point data.

Starting with the 4th generation Intel Xeon Scalable Processor Family based on Sapphire Rapids microarchitecture, Intel® AVX-512 instruction set architecture for FP16 was added, supporting a wide range of general-purpose numeric operations for 16-bit half precision floating-point values (binary16 in IEEE Standard 754-2019 for Floating-Point Arithmetic, aka half precision or FP16). Section 5.19 includes a list of these instructions.

5.15 FUSED-MULTIPLY-ADD (FMA)

FMA extensions enhances Intel AVX with high-throughput, arithmetic capabilities covering fused multiply-add, fused multiply-subtract, fused multiply add/subtract interleave, signed-reversed multiply on fused multiply-add and multiply-subtract. FMA extensions provide 36 256-bit floating-point instructions to perform computation on 256-bit vectors and additional 128-bit and scalar FMA instructions.

- Table 14-15 lists FMA instruction sets.

5.16 INTEL® ADVANCED VECTOR EXTENSIONS 2 (INTEL® AVX2)

Intel® AVX2 extends Intel AVX by promoting most of the 128-bit SIMD integer instructions with 256-bit numeric processing capabilities. Intel AVX2 instructions follow the same programming model as AVX instructions.

In addition, AVX2 provide enhanced functionalities for broadcast/permute operations on data elements, vector shift instructions with variable-shift count per data element, and instructions to fetch non-contiguous data elements from memory.

- Table 14-18 lists promoted vector integer instructions in AVX2.
- Table 14-19 lists new instructions in AVX2 that complements AVX.

5.17 INTEL® TRANSACTIONAL SYNCHRONIZATION EXTENSIONS (INTEL® TSX)

XABORT	Abort an RTM transaction execution.
XACQUIRE	Prefix hint to the beginning of an HLE transaction region.
XRELEASE	Prefix hint to the end of an HLE transaction region.
XBEGIN	Transaction begin of an RTM transaction region.
XEND	Transaction end of an RTM transaction region.
XTEST	Test if executing in a transactional region.
XRESLDRK	Resume tracking load addresses.
XSUSLDRK	Suspend tracking load addresses.

5.18 INTEL® SHA EXTENSIONS

Intel® SHA extensions provide a set of instructions that target the acceleration of the Secure Hash Algorithm (SHA), specifically the SHA-1 and SHA-256 variants.

SHA1MSG1	Perform an intermediate calculation for the next four SHA1 message dwords from the previous message dwords.
SHA1MSG2	Perform the final calculation for the next four SHA1 message dwords from the intermediate message dwords.
SHA1NEXTE	Calculate SHA1 state E after four rounds.
SHA1RND4	Perform four rounds of SHA1 operations.
SHA256MSG1	Perform an intermediate calculation for the next four SHA256 message dwords.
SHA256MSG2	Perform the final calculation for the next four SHA256 message dwords.
SHA256RND2	Perform two rounds of SHA256 operations.

5.19 INTEL® ADVANCED VECTOR EXTENSIONS 512 (INTEL® AVX-512)

The Intel® AVX-512 family comprises a collection of 512-bit SIMD instruction sets to accelerate a diverse range of applications. Intel AVX-512 instructions provide a wide range of functionality that support programming in 512-bit, 256 and 128-bit vector register, plus support for opmask registers and instructions operating on opmask registers.

The collection of 512-bit SIMD instruction sets in Intel AVX-512 include new functionality not available in Intel AVX and Intel AVX2, and promoted instructions similar to equivalent ones in Intel AVX/Intel AVX2 but with enhancement provided by opmask registers not available to VEX-encoded Intel AVX/Intel AVX2. Some instruction mnemonics in Intel AVX/Intel AVX2 that are promoted into Intel AVX-512 can be replaced by new instruction mnemonics that are available only with EVEX encoding, e.g., VBROADCASTF128 into VBROADCASTF32X4. Details of EVEX instruction encoding are discussed in Section 2.7, “Intel® AVX-512 Encoding,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A. Starting with the 4th generation Intel Xeon Scalable Processor Family, an Intel AVX-512 instruction set architecture for FP16 was added, supporting a wide range of

general-purpose numeric operations for 16-bit half precision floating-point values, which complements the existing 32-bit and 64-bit floating-point instructions already available in the Intel Xeon processor-based products.

512-bit instruction mnemonics in AVX-512F instructions that are not Intel AVX or AVX2 promotions include:

VALIGND/Q	Perform dword/qword alignment of two concatenated source vectors.
VBLENDMPD/PS	Replace the VBLENDVPD/PS instructions (using opmask as select control).
VCOMPRESSPD/PS	Compress packed DP or SP elements of a vector.
VCVT(T)PD2UDQ	Convert packed DP FP elements of a vector to packed unsigned 32-bit integers.
VCVT(T)PS2UDQ	Convert packed SP FP elements of a vector to packed unsigned 32-bit integers.
VCVTQQ2PD/PS	Convert packed signed 64-bit integers to packed DP/SP FP elements.
VCVT(T)SD2USI	Convert the low DP FP element of a vector to an unsigned integer.
VCVT(T)SS2USI	Convert the low SP FP element of a vector to an unsigned integer.
VCVTUDQ2PD/PS	Convert packed unsigned 32-bit integers to packed DP/SP FP elements.
VCVTUSI2USD/S	Convert an unsigned integer to the low DP/SP FP element and merge to a vector.
VEXPANDPD/PS	Expand packed DP or SP elements of a vector.
VEXTRACTF32X4/64X4	Extract a vector from a full-length vector with 32/64-bit granular update.
VEXTRACTI32X4/64X4	Extract a vector from a full-length vector with 32/64-bit granular update.
VFIXUPIMMPD/PS	Perform fix-up to special values in DP/SP FP vectors.
VFIXUPIMMSD/SS	Perform fix-up to special values of the low DP/SP FP element.
VGETEXPPD/PS	Convert the exponent of DP/SP FP elements of a vector into FP values.
VGETEXPSD/SS	Convert the exponent of the low DP/SP FP element in a vector into FP value.
VGETMANTPD/PS	Convert the mantissa of DP/SP FP elements of a vector into FP values.
VGETMANTSD/SS	Convert the mantissa of the low DP/SP FP element of a vector into FP value.
VINSERTF32X4/64X4	Insert a 128/256-bit vector into a full-length vector with 32/64-bit granular update.
VMOVDQA32/64	VMOVDQA with 32/64-bit granular conditional update.
VMOVDQU32/64	VMOVDQU with 32/64-bit granular conditional update.
VPBLENDMD/Q	Blend dword/qword elements using opmask as select control.
VPBROADCASTD/Q	Broadcast from general-purpose register to vector register.
VPCMPD/UD	Compare packed signed/unsigned dwords using specified primitive.
VPCMPQ/UQ	Compare packed signed/unsigned quadwords using specified primitive.
VPCOMPRESSQ/D	Compress packed 64/32-bit elements of a vector.
VPERMI2D/Q	Full permute of two tables of dword/qword elements overwriting the index vector.
VPERMI2PD/PS	Full permute of two tables of DP/SP elements overwriting the index vector.
VPERMT2D/Q	Full permute of two tables of dword/qword elements overwriting one source table.
VPERMT2PD/PS	Full permute of two tables of DP/SP elements overwriting one source table.
VPEXPANDD/Q	Expand packed dword/qword elements of a vector.
VPMAXSQ	Compute maximum of packed signed 64-bit integer elements.
VPMAXUD/UQ	Compute maximum of packed unsigned 32/64-bit integer elements.
VPMINSQ	Compute minimum of packed signed 64-bit integer elements.
VPMINUD/UQ	Compute minimum of packed unsigned 32/64-bit integer elements.
VPMOV(S US)QB	Down convert qword elements in a vector to byte elements using truncation (saturation unsigned saturation).
VPMOV(S US)QW	Down convert qword elements in a vector to word elements using truncation (saturation unsigned saturation).
VPMOV(S US)QD	Down convert qword elements in a vector to dword elements using truncation (saturation unsigned saturation).
VPMOV(S US)DB	Down convert dword elements in a vector to byte elements using truncation (saturation unsigned saturation).

VPMOV(S US)DW	Down convert dword elements in a vector to word elements using truncation (saturation unsigned saturation).
VPROLD/Q	Rotate dword/qword element left by a constant shift count with conditional update.
VPROLVD/Q	Rotate dword/qword element left by shift counts specified in a vector with conditional update.
VPRORD/Q	Rotate dword/qword element right by a constant shift count with conditional update.
VPRORRD/Q	Rotate dword/qword element right by shift counts specified in a vector with conditional update.
VPSCATTERDD/DQ	Scatter dword/qword elements in a vector to memory using dword indices.
VPSCATTERQD/QQ	Scatter dword/qword elements in a vector to memory using qword indices.
VPSRAQ	Shift qwords right by a constant shift count and shifting in sign bits.
VPSRAVQ	Shift qwords right by shift counts in a vector and shifting in sign bits.
VPTESTNMD/Q	Perform bitwise NAND of dword/qword elements of two vectors and write results to opmask.
VPTERLOGD/Q	Perform bitwise ternary logic operation of three vectors with 32/64 bit granular conditional update.
VPTESTMD/Q	Perform bitwise AND of dword/qword elements of two vectors and write results to opmask.
VRCP14PD/PS	Compute approximate reciprocals of packed DP/SP FP elements of a vector.
VRCP14SD/SS	Compute the approximate reciprocal of the low DP/SP FP element of a vector.
VRNDSCALEPD/PS	Round packed DP/SP FP elements of a vector to specified number of fraction bits.
VRNDSCALESD/SS	Round the low DP/SP FP element of a vector to specified number of fraction bits.
VRSQRT14PD/PS	Compute approximate reciprocals of square roots of packed DP/SP FP elements of a vector.
VRSQRT14SD/SS	Compute the approximate reciprocal of square root of the low DP/SP FP element of a vector.
VSCALEPD/PS	Multiply packed DP/SP FP elements of a vector by powers of two with exponents specified in a second vector.
VSCALESD/SS	Multiply the low DP/SP FP element of a vector by powers of two with exponent specified in the corresponding element of a second vector.
VSCATTERDD/DQ	Scatter SP/DP FP elements in a vector to memory using dword indices.
VSCATTERQD/QQ	Scatter SP/DP FP elements in a vector to memory using qword indices.
VSHUFF32X4/64X2	Shuffle 128-bit lanes of a vector with 32/64 bit granular conditional update.
VSHUFI32X4/64X2	Shuffle 128-bit lanes of a vector with 32/64 bit granular conditional update.

512-bit instruction mnemonics in AVX-512DQ that are not Intel AVX or AVX2 promotions include:

VCVT(T)PD2QQ	Convert packed DP FP elements of a vector to packed signed 64-bit integers.
VCVT(T)PD2UQQ	Convert packed DP FP elements of a vector to packed unsigned 64-bit integers.
VCVT(T)PS2QQ	Convert packed SP FP elements of a vector to packed signed 64-bit integers.
VCVT(T)PS2UQQ	Convert packed SP FP elements of a vector to packed unsigned 64-bit integers.
VCVTUQQ2PD/PS	Convert packed unsigned 64-bit integers to packed DP/SP FP elements.
VEXTRACTF64X2	Extract a vector from a full-length vector with 64-bit granular update.
VEXTRACTI64X2	Extract a vector from a full-length vector with 64-bit granular update.
VFPCLASSPD/PS	Test packed DP/SP FP elements in a vector by numeric/special-value category.
VFPCLASSSD/SS	Test the low DP/SP FP element by numeric/special-value category.
VINSERTF64X2	Insert a 128-bit vector into a full-length vector with 64-bit granular update.
VINSERTI64X2	Insert a 128-bit vector into a full-length vector with 64-bit granular update.
VPMOVM2D/Q	Convert opmask register to vector register in 32/64-bit granularity.
VPMOVB2D/Q2M	Convert a vector register in 32/64-bit granularity to an opmask register.

VPMULLQ	Multiply packed signed 64-bit integer elements of two vectors and store low 64-bit signed result.
VRANGEPS/PS	Perform RANGE operation on each pair of DP/SP FP elements of two vectors using specified range primitive in imm8.
VRANGESD/SS	Perform RANGE operation on the pair of low DP/SP FP element of two vectors using specified range primitive in imm8.
VREDUCEPS/PS	Perform Reduction operation on packed DP/SP FP elements of a vector using specified reduction primitive in imm8.
VREDUCESD/SS	Perform Reduction operation on the low DP/SP FP element of a vector using specified reduction primitive in imm8.

512-bit instruction mnemonics in AVX-512BW that are not Intel AVX or AVX2 promotions include:

VDBPSADBW	Double block packed Sum-Absolute-Differences on unsigned bytes.
VMOVDQU8/16	VMOVDQU with 8/16-bit granular conditional update.
VPBLENDMB	Replaces the VPBLENDVB instruction (using opmask as select control).
VPBLENDMW	Blend word elements using opmask as select control.
VPBROADCASTB/W	Broadcast from general-purpose register to vector register.
VPCMPB/UB	Compare packed signed/unsigned bytes using specified primitive.
VPCMPW/UW	Compare packed signed/unsigned words using specified primitive.
VPERMW	Permute packed word elements.
VPERMI2W	Full permute from two tables of word elements overwriting the index vector.
VPMOVM2B/W	Convert opmask register to vector register in 8/16-bit granularity.
VPMOVB2M/W2M	Convert a vector register in 8/16-bit granularity to an opmask register.
VPMOV(S US)WB	Down convert word elements in a vector to byte elements using truncation (saturation unsigned saturation).
VPSLLVW	Shift word elements in a vector left by shift counts in a vector.
VPSRAVW	Shift words right by shift counts in a vector and shifting in sign bits.
VPSRLVW	Shift word elements in a vector right by shift counts in a vector.
VPTESTNMB/W	Perform bitwise NAND of byte/word elements of two vectors and write results to opmask.
VPTESTMB/W	Perform bitwise AND of byte/word elements of two vectors and write results to opmask.

512-bit instruction mnemonics in AVX-512CD that are not Intel AVX or AVX2 promotions include:

VPBROADCASTM	Broadcast from opmask register to vector register.
VPCONFLICTD/Q	Detect conflicts within a vector of packed 32/64-bit integers.
VPLZCNTD/Q	Count the number of leading zero bits of packed dword/qword elements.

Opmask instructions include:

KADDB/W/D/Q	Add two 8/16/32/64-bit opmasks.
KANDB/W/D/Q	Logical AND two 8/16/32/64-bit opmasks.
KANDNB/W/D/Q	Logical AND NOT two 8/16/32/64-bit opmasks.
KMOVB/W/D/Q	Move from or move to opmask register of 8/16/32/64-bit data.
KNOTB/W/D/Q	Bitwise NOT of two 8/16/32/64-bit opmasks.
KORB/W/D/Q	Logical OR two 8/16/32/64-bit opmasks.
KORTESTB/W/D/Q	Update EFLAGS according to the result of bitwise OR of two 8/16/32/64-bit opmasks.
KSHIFTLB/W/D/Q	Shift left 8/16/32/64-bit opmask by specified count.
KSHIFTRB/W/D/Q	Shift right 8/16/32/64-bit opmask by specified count.
KTESTB/W/D/Q	Update EFLAGS according to the result of bitwise TEST of two 8/16/32/64-bit opmasks.

KUNPCKBW/WD/DQ	Unpack and interleave two 8/16/32-bit opmasks into 16/32/64-bit mask.
KXNORB/W/D/Q	Bitwise logical XNOR of two 8/16/32/64-bit opmasks.
KXORB/W/D/Q	Logical XOR of two 8/16/32/64-bit opmasks.

512-bit instruction mnemonics in AVX-512ER include:

VEXP2PD/PS	Compute approximate base-2 exponential of packed DP/SP FP elements of a vector.
VEXP2SD/SS	Compute approximate base-2 exponential of the low DP/SP FP element of a vector.
VRCP28PD/PS	Compute approximate reciprocals to 28 bits of packed DP/SP FP elements of a vector.
VRCP28SD/SS	Compute the approximate reciprocal to 28 bits of the low DP/SP FP element of a vector.
VRSQRT28PD/PS	Compute approximate reciprocals of square roots to 28 bits of packed DP/SP FP elements of a vector.
VRSQRT28SD/SS	Compute the approximate reciprocal of square root to 28 bits of the low DP/SP FP element of a vector.

512-bit instruction mnemonics in AVX-512PF include:

VGATHERPF0DPD/PS	Sparse prefetch of packed DP/SP FP vector with T0 hint using dword indices.
VGATHERPF0QPD/PS	Sparse prefetch of packed DP/SP FP vector with T0 hint using qword indices.
VGATHERPF1DPD/PS	Sparse prefetch of packed DP/SP FP vector with T1 hint using dword indices.
VGATHERPF1QPD/PS	Sparse prefetch of packed DP/SP FP vector with T1 hint using qword indices.
VSCATTERPF0DPD/PS	Sparse prefetch of packed DP/SP FP vector with T0 hint to write using dword indices.
VSCATTERPF0QPD/PS	Sparse prefetch of packed DP/SP FP vector with T0 hint to write using qword indices.
VSCATTERPF1DPD/PS	Sparse prefetch of packed DP/SP FP vector with T1 hint to write using dword indices.
VSCATTERPF1QPD/PS	Sparse prefetch of packed DP/SP FP vector with T1 hint to write using qword indices.

512-bit instruction mnemonics in AVX512-FP16 include:

VADDPH/SH	Add packed/scalar FP16 values.
VCMPPH/SH	Compare packed/scalar FP16 values.
VCOMISH	Compare scalar ordered FP16 values and set EFLAGS.
VCVTDQ2PH	Convert packed signed doubleword integers to packed FP16 values.
VCVTPD2PH	Convert packed double precision FP values to packed FP16 values.
VCVTPH2DQ/QQ	Convert packed FP16 values to signed doubleword/quadword integers.
VCVTPH2PD	Convert packed FP16 values to FP64 values.
VCVTPH2PS[X]	Convert packed FP16 values to single precision floating-point values.
VCVTPH2QQ	Convert packed FP16 values to signed quadword integer values.
VCVTPH2UDQ/QQ	Convert packed FP16 values to unsigned doubleword/quadword integers.
VCVTPH2UW/W	Convert packed FP16 values to unsigned/signed word integers.
VCVTPS2PH[X]	Convert packed single precision floating-point values to packed FP16 values.
VCVTQQ2PH	Convert packed signed quadword integers to packed FP16 values.
VCVTSD2SH	Convert low FP64 value to an FP16 value.
VCVTSH2SD/SS	Convert low FP16 value to an FP64/FP32 value.
VCVTSH2SI/USI	Convert low FP16 value to signed/unsigned integer.
VCVTSI2SH	Convert a signed doubleword/quadword integer to an FP16 value.
VCVTSS2SH	Convert low FP32 value to an FP16 value.
VCVTTPH2DQ/QQ	Convert with truncation packed FP16 values to signed doubleword/quadword integers.
VCVTTPH2UDQ/QQ	Convert with truncation packed FP16 values to unsigned doubleword/quadword integers.
VCVTTPH2UW/W	Convert packed FP16 values to unsigned/signed word integers.

VCVTTSH2SI/USI	Convert with truncation low FP16 value to a signed/unsigned integer.
VCVTUDQ2PH	Convert packed unsigned doubleword integers to packed FP16 values.
VCVTUQQ2PH	Convert packed unsigned quadword integers to packed FP16 values.
VCVTUSI2SH	Convert unsigned doubleword integer to an FP16 value.
VCVTUW2PH	Convert packed unsigned word integers to FP16 values.
VCVTW2PH	Convert packed signed word integers to FP16 values.
VDIVPH/SH	Divide packed/scalar FP16 values.
VF[C]MADDCPH	Complex multiply and accumulate FP16 values.
VF[C]MADDCSH	Complex multiply and accumulate scalar FP16 values.
VF[C]MULCPH	Complex multiply FP16 values.
VF[C]MULCSH	Complex multiply scalar FP16 values.
VF[,N]MADD[132,213,231]PH	Fused multiply-add of packed FP16 values.
VF[,N]MADD[132,213,231]SH	Fused multiply-add of scalar FP16 values.
VFMADDSUB[132,213,231]PH	Fused multiply-alternating add/subtract of packed FP16 values.
VFMSUBADD[132,213,231]PH	Fused multiply-alternating subtract/add of packed FP16 values.
VF[,N]MSUB[132,213,231]PH	Fused multiply-subtract of packed FP16 values.
VF[,N]MSUB[132,213,231]SH	Fused multiply-subtract of scalar FP16 values.
VFPCCLASSPH/SH	Test types of packed/scalar FP16 values.
VGETEXPPH/SH	Convert exponents of packed/scalar FP16 values to FP16 values.
VGETMANTPH/SH	Extract FP16 vector of normalized mantissas from FP16 vector/scalar.
VMAXPH/PS	Return maximum of packed/scalar FP16 values.
VMINPH/PS	Return minimum of packed/scalar FP16 values.
VMOVSH	Move scalar FP16 value.
VMOVW	Move word.
VMULPH/SH	Multiply packed/scalar FP16 values.
VRCPPH/SH	Compute reciprocals of packed/scalar FP16 values.
VREDUCEPH/SH	Perform reduction transformation on packed/scalar FP16 values.
VRNDSCALEPH/SH	Round packed/scalar FP16 values to include a given number of fraction bits.
VRSQRTPH/SH	Compute reciprocals of square roots of packed/scalar FP16 values.
VSCALEPH/SH	Scale packed/scalar FP16 values with FP16 values.
VSQRTPH/SH	Compute square root of packed/scalar FP16 values.
VSUBPH/SH	Subtract packed/scalar FP16 values.
VUCOMISH	Unordered compare scalar FP16 values and set EFLAGS.

5.20 SYSTEM INSTRUCTIONS

The following system instructions are used to control those functions of the processor that are provided to support for operating systems and executives.

CLAC	Clear AC Flag in EFLAGS register.
STAC	Set AC Flag in EFLAGS register.
LGDT	Load global descriptor table (GDT) register.
SGDT	Store global descriptor table (GDT) register.
LLDT	Load local descriptor table (LDT) register.
SLDT	Store local descriptor table (LDT) register.
LTR	Load task register.
STR	Store task register.

LIDT	Load interrupt descriptor table (IDT) register.
SIDT	Store interrupt descriptor table (IDT) register.
MOV	Load and store control registers.
LMSW	Load machine status word.
SMSW	Store machine status word.
CLTS	Clear the task-switched flag.
ARPL	Adjust requested privilege level.
LAR	Load access rights.
LSL	Load segment limit.
VERR	Verify segment for reading
VERW	Verify segment for writing.
MOV	Load and store debug registers.
INVD	Invalidate cache, no writeback.
WBINVD	Invalidate cache, with writeback.
INVLPG	Invalidate TLB Entry.
INVPID	Invalidate Process-Context Identifier.
LOCK (prefix)	Perform atomic access to memory (can be applied to a number of general purpose instructions that provide memory source/destination access).
HLT	Halt processor.
RSM	Return from system management mode (SMM).
RDMSR	Read model-specific register.
WRMSR	Write model-specific register.
RDPMC	Read performance monitoring counters.
RDTSC	Read time stamp counter.
RDTSCP	Read time stamp counter and processor ID.
SYSENTER	Fast System Call, transfers to a flat protected mode kernel at CPL = 0.
SYSEXIT	Fast System Call, transfers to a flat protected mode kernel at CPL = 3.
XSAVE	Save processor extended states to memory.
XSAVEC	Save processor extended states with compaction to memory.
XSAVEOPT	Save processor extended states to memory, optimized.
XSAVES	Save processor supervisor-mode extended states to memory.
XRSTOR	Restore processor extended states from memory.
XRSTORS	Restore processor supervisor-mode extended states from memory.
XGETBV	Reads the state of an extended control register.
XSETBV	Writes the state of an extended control register.
RDFSBASE	Reads from FS base address at any privilege level.
RDGSBASE	Reads from GS base address at any privilege level.
WRFSBASE	Writes to FS base address at any privilege level.
WRGSBASE	Writes to GS base address at any privilege level.

5.21 64-BIT MODE INSTRUCTIONS

The following instructions are introduced in 64-bit mode. This mode is a sub-mode of IA-32e mode.

CDQE	Convert doubleword to quadword.
CMPSQ	Compare string operands.
CMPXCHG16B	Compare RDX:RAX with m128.

LODSQ	Load qword at address (R)SI into RAX.
MOVSQ	Move qword from address (R)SI to (R)DI.
MOVZX (64-bits)	Move bytes/words to doublewords/quadwords, zero-extension.
STOSQ	Store RAX at address RDI.
SWAPGS	Exchanges current GS base register value with value in MSR address C0000102H.
SYSCALL	Fast call to privilege level 0 system procedures.
SYSRET	Return from fast system call.

5.22 VIRTUAL-MACHINE EXTENSIONS

The behavior of the VMCS-maintenance instructions is summarized below:

VMPTRLD	Takes a single 64-bit source operand in memory. It makes the referenced VMCS active and current.
VMPTRST	Takes a single 64-bit destination operand that is in memory. Current-VMCS pointer is stored into the destination operand.
VMCLEAR	Takes a single 64-bit operand in memory. The instruction sets the launch state of the VMCS referenced by the operand to “clear”, renders that VMCS inactive, and ensures that data for the VMCS have been written to the VMCS-data area in the referenced VMCS region.
VMREAD	Reads a component from the VMCS (the encoding of that field is given in a register operand) and stores it into a destination operand.
VMWRITE	Writes a component to the VMCS (the encoding of that field is given in a register operand) from a source operand.

The behavior of the VMX management instructions is summarized below:

VMLAUNCH	Launches a virtual machine managed by the VMCS. A VM entry occurs, transferring control to the VM.
VMRESUME	Resumes a virtual machine managed by the VMCS. A VM entry occurs, transferring control to the VM.
VMXOFF	Causes the processor to leave VMX operation.
VMXON	Takes a single 64-bit source operand in memory. It causes a logical processor to enter VMX root operation and to use the memory referenced by the operand to support VMX operation.

The behavior of the VMX-specific TLB-management instructions is summarized below:

INVEPT	Invalidate cached Extended Page Table (EPT) mappings in the processor to synchronize address translation in virtual machines with memory-resident EPT pages.
INVVPID	Invalidate cached mappings of address translation based on the Virtual Processor ID (VPID).

None of the instructions above can be executed in compatibility mode; they generate invalid-opcode exceptions if executed in compatibility mode.

The behavior of the guest-available instructions is summarized below:

VMCALL	Allows a guest in VMX non-root operation to call the VMM for service. A VM exit occurs, transferring control to the VMM.
VMFUNC	Allows software in VMX non-root operation to invoke a VM function, which is processor functionality enabled and configured by software in VMX root operation. No VM exit occurs.

5.23 SAFER MODE EXTENSIONS

The behavior of the GETSEC instruction leaves of the Safer Mode Extensions (SMX) are summarized below:

GETSEC[CAPABILITIES]	Returns the available leaf functions of the GETSEC instruction.
GETSEC[ENTERACCS]	Loads an authenticated code chipset module and enters authenticated code execution mode.
GETSEC[EXITAC]	Exits authenticated code execution mode.
GETSEC[SENDER]	Establishes a Measured Launched Environment (MLE) which has its dynamic root of trust anchored to a chipset supporting Intel Trusted Execution Technology.
GETSEC[SEXIT]	Exits the MLE.
GETSEC[PARAMETERS]	Returns SMX related parameter information.
GETSEC[SMCTRL]	SMX mode control.
GETSEC[WAKEUP]	Wakes up sleeping logical processors inside an MLE.

5.24 INTEL® MEMORY PROTECTION EXTENSIONS

Intel Memory Protection Extensions (Intel MPX) provides a set of instructions to enable software to add robust bounds checking capability to memory references. Details of Intel MPX are described in Appendix E, “Intel® Memory Protection Extensions.”

BNDMK	Create a LowerBound and an UpperBound in a register.
BNDCL	Check the address of a memory reference against a LowerBound.
BNDCU	Check the address of a memory reference against an UpperBound in 1’s complement form.
BNDCN	Check the address of a memory reference against an UpperBound not in 1’s complement form.
BNDMOV	Copy or load from memory of the LowerBound and UpperBound to a register.
BNDMOV	Store to memory of the LowerBound and UpperBound from a register.
BNDLDX	Load bounds using address translation.
BNDSTX	Store bounds using address translation.

5.25 INTEL® SOFTWARE GUARD EXTENSIONS

Intel Software Guard Extensions (Intel SGX) provide two sets of instruction leaf functions to enable application software to instantiate a protected container, referred to as an enclave. The enclave instructions are organized as leaf functions under two instruction mnemonics: ENCLS (ring 0) and ENCLU (ring 3). Details of Intel SGX are described in Chapter 37 through Chapter 43 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D.

The first implementation of Intel SGX is also referred to as SGX1, it is introduced with the 6th Generation Intel Core Processors. The leaf functions supported in SGX1 are shown in Table 5-3.

Table 5-3. Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX1

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EADD]	Add a page	ENCLU[EENTER]	Enter an Enclave
ENCLS[EBLOCK]	Block an EPC page	ENCLU[EEXIT]	Exit an Enclave
ENCLS[ECREATE]	Create an enclave	ENCLU[EGETKEY]	Create a cryptographic key
ENCLS[EDBGGRD]	Read data by debugger	ENCLU[EREPORT]	Create a cryptographic report
ENCLS[EDBGWR]	Write data by debugger	ENCLU[ERESUME]	Re-enter an Enclave
ENCLS[EEXTEND]	Extend EPC page measurement		
ENCLS[EINIT]	Initialize an enclave		
ENCLS[ELDB]	Load an EPC page as blocked		
ENCLS[ELDU]	Load an EPC page as unblocked		

Table 5-3. Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX1

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EPA]	Add version array		
ENCLS[EREMOVE]	Remove a page from EPC		
ENCLS[ETRACK]	Activate EBLOCK checks		
ENCLS[EWB]	Write back/invalidate an EPC page		

5.26 SHADOW STACK MANAGEMENT INSTRUCTIONS

Shadow stack management instructions allow the program and run-time to perform operations like recovering from control protection faults, shadow stack switching, etc. The following instructions are provided.

CLRSSBSY	Clear busy bit in a supervisor shadow stack token.
INCSSP	Increment the shadow stack pointer (SSP).
RDSSP	Read shadow stack point (SSP).
RSTORSSP	Restore a shadow stack pointer (SSP).
SAVEPREVSSP	Save previous shadow stack pointer (SSP).
SETSSBSY	Set busy bit in a supervisor shadow stack token.
WRSS	Write to a shadow stack.
WRUSS	Write to a user mode shadow stack.

5.27 CONTROL TRANSFER TERMINATING INSTRUCTIONS

ENDBR32	Terminate an Indirect Branch in 32-bit and Compatibility Mode.
ENDBR64	Terminate an Indirect Branch in 64-bit Mode.

5.28 INTEL® AMX INSTRUCTIONS

LDTILECFG	Load tile configuration.
STTILECFG	Store tile configuration.
TDPBF16PS	Dot product of BF16 tiles accumulated into packed single precision tile.
TDPBSSD	Dot product of signed bytes with dword accumulation.
TDPBSUD	Dot product of signed/unsigned bytes with dword accumulation.
TDPBUSD	Dot product of unsigned/signed bytes with dword accumulation.
TDPBUUD	Dot product of unsigned bytes with dword accumulation.
TILELOADD	Load data into tile.
TILELOADDT1	Load data into tile with hint to optimize data caching.
TILERELASE	Release tile.
TILESTORED	Store tile.
TILEZERO	Zero tile.

5.29 USER INTERRUPT INSTRUCTIONS

CLUI	Clear user interrupt flag.
SENDUIPI	Send user interprocessor interrupt.
STUI	Set user interrupt flag.

TESTUI	Determine user interrupt flag.
UIRET	User-interrupt return.

5.30 ENQUEUE STORE INSTRUCTIONS

ENQCMD	Enqueue command.
ENQCMD5	Enqueue command supervisor.

5.31 INTEL® ADVANCED VECTOR EXTENSIONS 10 VERSION 1 INSTRUCTIONS

Intel® Advanced Vector Extensions 10 Version 1 (Intel® AVX10.1) is based on the Intel AVX-512 ISA feature set and includes all Intel AVX-512 instructions introduced with the Intel® Xeon® 6 P-core processor based on Granite Rapids microarchitecture. Intel AVX10.1 supports all instruction vector lengths (128, 256, and 512), as well as scalar and opmask instructions.

For a list of Intel AVX-512 instructions, see Section 5.19, “Intel® Advanced Vector Extensions 512 (Intel® AVX-512).” Additionally, note that some Intel AVX and Intel AVX2 instructions were promoted to Intel AVX512 and are also supported. See Section 5.13, “Intel® Advanced Vector Extensions (Intel® AVX),” Section 5.16, “Intel® Advanced Vector Extensions 2 (Intel® AVX2),” and Chapter 16, “Programming with Intel® AVX10,” for further details.

2. Updates to Chapter 6, Volume 1

Change bars and text show changes to Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

Changes to this chapter:

- Added Flexible Return and Event Delivery (FRED) references that define new mechanisms for changing privilege levels in Section 6.4.5, "Calls to Other Privilege Levels".
- In Section 6.5, Section 6.5.1, and Section 6.5.6, documented how interrupts and exceptions are handled with FRED.

CHAPTER 6

PROCEDURE CALLS, INTERRUPTS, AND EXCEPTIONS

This chapter describes the facilities in the Intel 64 and IA-32 architectures for executing calls to procedures or subroutines. It also describes how interrupts and exceptions are handled from the perspective of an application programmer.

6.1 PROCEDURE CALL TYPES

The processor supports procedure calls in the following two different ways:

- CALL and RET instructions.
- ENTER and LEAVE instructions, in conjunction with the CALL and RET instructions.

Both of these procedure call mechanisms use the procedure stack, commonly referred to simply as “the stack,” to save the state of the calling procedure, pass parameters to the called procedure, and store local variables for the currently executing procedure.

The processor’s facilities for handling interrupts and exceptions are similar to those used by the CALL and RET instructions.

Processors that support Control-Flow Enforcement Technology (CET) support an additional stack referred to as “the shadow stack”. The CALL instruction, when shadow stacks are enabled, additionally saves the state of the calling procedure on the shadow stack; and the RET instruction restores the state of the calling procedure if the state on the stack and the shadow stack match.

6.2 STACKS

The stack (see Figure 6-1) is a contiguous array of memory locations. It is contained in a segment and identified by the segment selector in the SS register. When using the flat memory model, the stack can be located anywhere in the linear address space for the program. A stack can be up to 4 GBytes long, the maximum size of a segment.

Items are placed on the stack using the PUSH instruction and removed from the stack using the POP instruction. When an item is pushed onto the stack, the processor decrements the ESP register, then writes the item at the new top of stack. When an item is popped off the stack, the processor reads the item from the top of stack, then increments the ESP register. In this manner, the stack grows **down** in memory (towards lesser addresses) when items are pushed on the stack and shrinks **up** (towards greater addresses) when the items are popped from the stack.

A program or operating system/executive can set up many stacks. For example, in multitasking systems, each task can be given its own stack. The number of stacks in a system is limited by the maximum number of segments and the available physical memory.

When a system sets up many stacks, only one stack—the **current stack**—is available at a time. The current stack is the one contained in the segment referenced by the SS register.

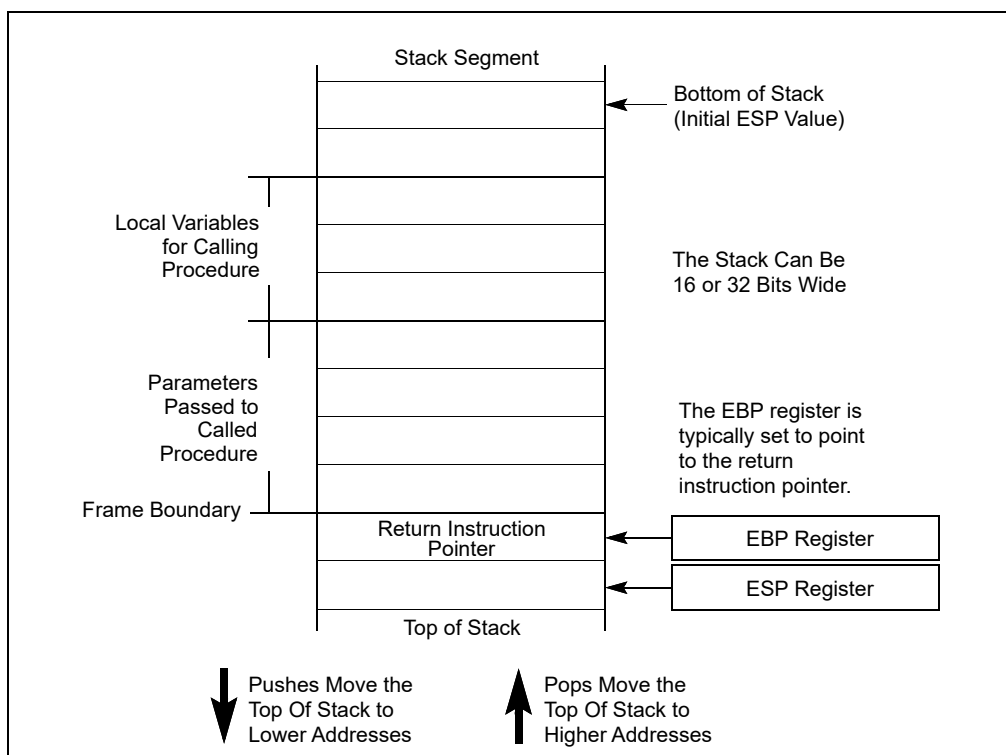


Figure 6-1. Stack Structure

The processor references the SS register automatically for all stack operations. For example, when the ESP register is used as a memory address, it automatically points to an address in the current stack. Also, the CALL, RET, PUSH, POP, ENTER, and LEAVE instructions all perform operations on the current stack.

6.2.1 Setting Up a Stack

To set a stack and establish it as the current stack, the program or operating system/executive must do the following:

1. Establish a stack segment.
2. Load the segment selector for the stack segment into the SS register using a MOV, POP, or LSS instruction.
3. Load the stack pointer for the stack into the ESP register using a MOV, POP, or LSS instruction. The LSS instruction can be used to load the SS and ESP registers in one operation.

See "Segment Descriptors" in Chapter 3, "Protected-Mode Memory Management," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for information on how to set up a segment descriptor and segment limits for a stack segment.

6.2.2 Stack Alignment

The stack pointer for a stack segment should be aligned on 16-bit (word) or 32-bit (double-word) boundaries, depending on the width of the stack segment. The D flag in the segment descriptor for the current code segment sets the stack-segment width (see "Segment Descriptors" in Chapter 3, "Protected-Mode Memory Management," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A). The PUSH and POP instructions use the D flag to determine how much to decrement or increment the stack pointer on a push or pop operation, respectively. When the stack width is 16 bits, the stack pointer is incremented or decremented in 16-bit increments; when the width is 32 bits, the stack pointer is incremented or decremented in 32-bit increments. Pushing a 16-bit value onto a 32-bit wide stack can result in stack misaligned (that is, the stack pointer is not aligned on a double-

word boundary). One exception to this rule is when the contents of a segment register (a 16-bit segment selector) are pushed onto a 32-bit wide stack. Here, the processor automatically aligns the stack pointer to the next 32-bit boundary.

The processor does not check stack pointer alignment. It is the responsibility of the programs, tasks, and system procedures running on the processor to maintain proper alignment of stack pointers. Misaligning a stack pointer can cause serious performance degradation and in some instances program failures.

6.2.3 Address-Size Attributes for Stack Accesses

Instructions that use the stack implicitly (such as the PUSH and POP instructions) have two address-size attributes each of either 16 or 32 bits. This is because they always have the implicit address of the top of the stack, and they may also have an explicit memory address (for example, PUSH Array1[EBX]). The attribute of the explicit address is determined by the D flag of the current code segment and the presence or absence of the 67H address-size prefix.

The address-size attribute of the top of the stack determines whether SP or ESP is used for the stack access. Stack operations with an address-size attribute of 16 use the 16-bit SP stack pointer register and can use a maximum stack address of FFFFH; stack operations with an address-size attribute of 32 bits use the 32-bit ESP register and can use a maximum address of FFFFFFFFH. The default address-size attribute for data segments used as stacks is controlled by the B flag of the segment's descriptor. When this flag is clear, the default address-size attribute is 16; when the flag is set, the address-size attribute is 32.

6.2.4 Procedure Linking Information

The processor provides two pointers for linking of procedures: the stack-frame base pointer and the return instruction pointer. When used in conjunction with a standard software procedure-call technique, these pointers permit reliable and coherent linking of procedures.

6.2.4.1 Stack-Frame Base Pointer

The stack is typically divided into frames. Each stack frame can then contain local variables, parameters to be passed to another procedure, and procedure linking information. The stack-frame base pointer (contained in the EBP register) identifies a fixed reference point within the stack frame for the called procedure. To use the stack-frame base pointer, the called procedure typically copies the contents of the ESP register into the EBP register prior to pushing any local variables on the stack. The stack-frame base pointer then permits easy access to data structures passed on the stack, to the return instruction pointer, and to local variables added to the stack by the called procedure.

Like the ESP register, the EBP register automatically points to an address in the current stack segment (that is, the segment specified by the current contents of the SS register).

6.2.4.2 Return Instruction Pointer

Prior to branching to the first instruction of the called procedure, the CALL instruction pushes the address in the EIP register onto the current stack. This address is then called the return-instruction pointer and it points to the instruction where execution of the calling procedure should resume following a return from the called procedure. Upon returning from a called procedure, the RET instruction pops the return-instruction pointer from the stack back into the EIP register. Execution of the calling procedure then resumes.

The processor does not keep track of the location of the return-instruction pointer. It is thus up to the programmer to ensure that stack pointer is pointing to the return-instruction pointer on the stack, prior to issuing a RET instruction. A common way to reset the stack pointer to the point to the return-instruction pointer is to move the contents of the EBP register into the ESP register. If the EBP register is loaded with the stack pointer immediately following a procedure call, it should point to the return instruction pointer on the stack.

The processor does not require that the return instruction pointer point back to the calling procedure. Prior to executing the RET instruction, the return instruction pointer can be manipulated in software to point to any address

in the current code segment (near return) or another code segment (far return). Performing such an operation, however, should be undertaken very cautiously, using only well defined code entry points.

6.2.5 Stack Behavior in 64-Bit Mode

In 64-bit mode, address calculations that reference SS segments are treated as if the segment base is zero. Fields (base, limit, and attribute) in segment descriptor registers are ignored. SS DPL is modified such that it is always equal to CPL. This will be true even if it is the only field in the SS descriptor that is modified.

Registers E(SP), E(IP) and E(BP) are promoted to 64-bits and are re-named RSP, RIP, and RBP respectively. Some forms of segment load instructions are invalid (for example, LDS, POP ES).

PUSH/POP instructions increment/decrement the stack using a 64-bit width. When the contents of a segment register is pushed onto 64-bit stack, the pointer is automatically aligned to 64 bits (as with a stack that has a 32-bit width).

6.3 SHADOW STACKS

A shadow stack is a second stack used exclusively for control transfer operations. This stack is separate from the procedure stack. The shadow stack is not used to store data, hence is not explicitly writeable by software. Writes to the shadow stack are restricted to control transfer instructions and shadow stack management instructions. Shadow stacks can be enabled separately for privilege level 3 (user mode) or privilege levels less than 3 (supervisor mode).

Shadow stacks are active only in protected mode with paging enabled. Shadow stacks cannot be enabled for a program executing in virtual 8086 mode.

Processors that support shadow stacks have an architectural register called the shadow stack pointer (SSP) that points to the current top of the shadow stack. The SSP cannot be directly encoded as a source, destination, or memory operand in instructions. The width of the shadow stack is 32-bit in 32-bit/compatibility mode, and is 64-bit in 64-bit mode. The address-size attribute of the shadow stack is likewise 32-bit in 32-bit/compatibility mode, and 64-bit in 64-bit mode.

The size of the shadow stack pushes and pops for far CALL and call to interrupt/exception handlers is fixed at 64 bits, and the processor uses 8-byte, zero padded stores for these pushes in 32-bit/compatibility modes.

6.4 CALLING PROCEDURES USING CALL AND RET

The CALL instruction allows control transfers to procedures within the current code segment (**near call**) and in a different code segment (**far call**). Near calls usually provide access to local procedures within the currently running program or task. Far calls are usually used to access operating system procedures or procedures in a different task. See "CALL—Call Procedure" in Chapter 3, "Instruction Set Reference, A-L," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for a detailed description of the CALL instruction.

The RET instruction also allows near and far returns to match the near and far versions of the CALL instruction. In addition, the RET instruction allows a program to increment the stack pointer on a return to release parameters from the stack. The number of bytes released from the stack is determined by an optional argument (*n*) to the RET instruction. See "RET—Return from Procedure" in Chapter 4, "Instruction Set Reference, M-U," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B, for a detailed description of the RET instruction.

6.4.1 Near CALL and RET Operation

When executing a near call, the processor does the following (see Figure 6-2):

1. Pushes the current value of the EIP register on the stack.

If shadow stack is enabled and the displacement value is not 0, pushes the current value of the EIP register on the shadow stack.

2. Loads the offset of the called procedure in the EIP register.
3. Begins execution of the called procedure.

When executing a near return, the processor performs these actions:

1. Pops the top-of-stack value (the return instruction pointer) into the EIP register.
If shadow stack is enabled, pops the top-of-stack (the return instruction pointer) value from the shadow stack and if it's not the same as the return instruction pointer popped from the stack, then the processor causes a control protection exception with error code NEAR-RET (#CP(NEAR-RET)).
2. If the RET instruction has an optional *n* argument, increments the stack pointer by the number of bytes specified with the *n* operand to release parameters from the stack.
3. Resumes execution of the calling procedure.

6.4.2 Far CALL and RET Operation

When executing a far call, the processor performs these actions (see Figure 6-2):

1. Pushes the current value of the CS register on the stack.
If shadow stack is enabled:
 - a. Temporarily saves the current value of the SSP register internally and aligns the SSP to the next 8 byte boundary.
 - b. Pushes the current value of the CS register on the shadow stack.
 - c. Pushes the current value of LIP (CS.base + EIP) on the shadow stack.
 - d. Pushes the internally saved value of the SSP register on the shadow stack.
2. Pushes the current value of the EIP register on the stack.
3. Loads the segment selector of the segment that contains the called procedure in the CS register.
4. Loads the offset of the called procedure in the EIP register.
5. Begins execution of the called procedure.

When executing a far return, the processor does the following:

1. Pops the top-of-stack value (the return instruction pointer) into the EIP register.
2. Pops the top-of-stack value (the segment selector for the code segment being returned to) into the CS register.
If shadow stack is enabled:
 - a. Causes a control protection exception (#CP(FAR-RET/IRET)) if the SSP is not aligned to 8 bytes.
 - b. Compares the values on the shadow stack at address SSP+8 (the LIP) and SSP+16 (the CS) to the CS and (CS.base + EIP) popped from the stack, and causes a control protection exception (#CP(FAR-RET/IRET)) if they do not match.
 - c. Pops the top-of-stack value (the SSP of the procedure being returned to) from shadow stack into the SSP register.
3. If the RET instruction has an optional *n* argument, increments the stack pointer by the number of bytes specified with the *n* operand to release parameters from the stack.
4. Resumes execution of the calling procedure.

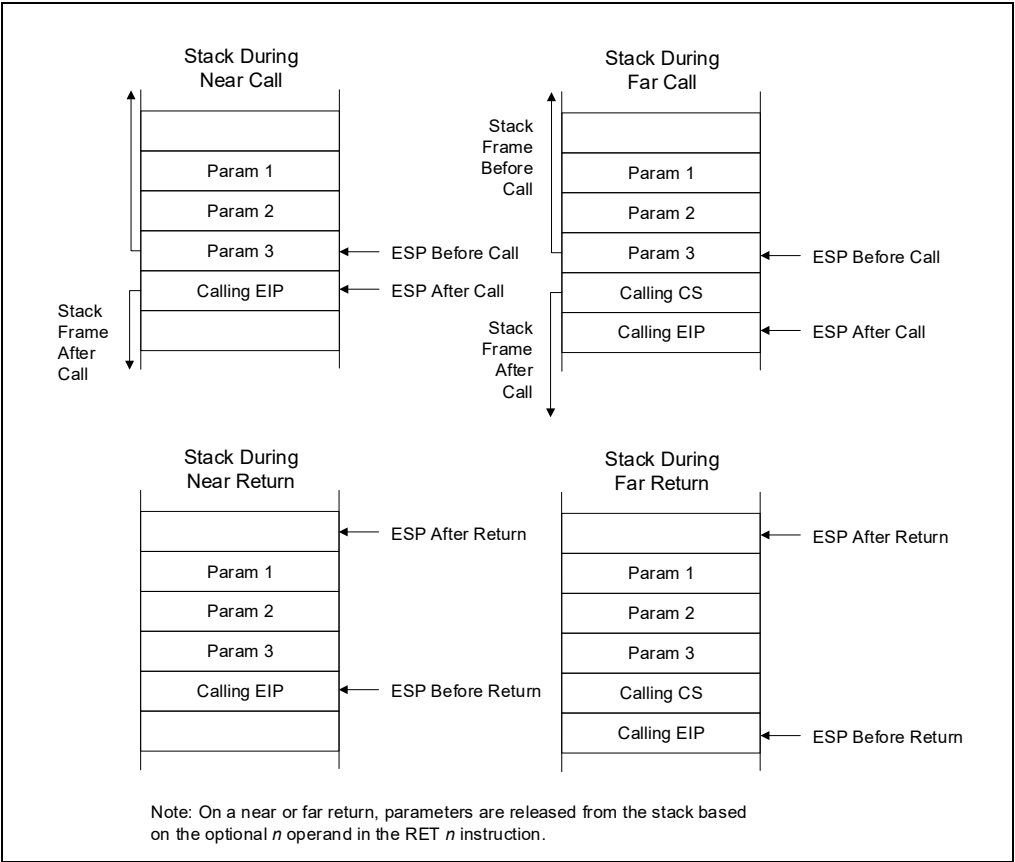


Figure 6-2. Stack on Near and Far Calls

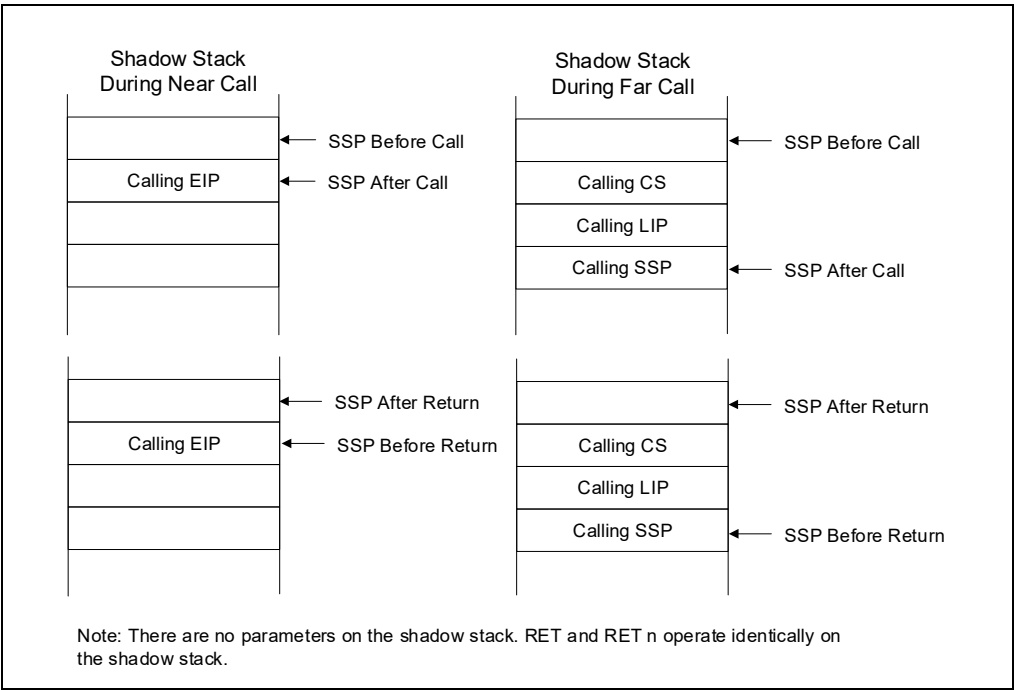


Figure 6-3. Shadow Stack on Near and Far Calls

6.4.3 Parameter Passing

Parameters can be passed between procedures in any of three ways: through general-purpose registers, in an argument list, or on the stack.

6.4.3.1 Passing Parameters Through the General-Purpose Registers

The processor does not save the state of the general-purpose registers on procedure calls. A calling procedure can thus pass up to six parameters to the called procedure by copying the parameters into any of these registers (except the ESP and EBP registers) prior to executing the CALL instruction. The called procedure can likewise pass parameters back to the calling procedure through general-purpose registers.

6.4.3.2 Passing Parameters on the Stack

To pass a large number of parameters to the called procedure, the parameters can be placed on the stack, in the stack frame for the calling procedure. Here, it is useful to use the stack-frame base pointer (in the EBP register) to make a frame boundary for easy access to the parameters.

The stack can also be used to pass parameters back from the called procedure to the calling procedure.

6.4.3.3 Passing Parameters in an Argument List

An alternate method of passing a larger number of parameters (or a data structure) to the called procedure is to place the parameters in an argument list in one of the data segments in memory. A pointer to the argument list can then be passed to the called procedure through a general-purpose register or the stack. Parameters can also be passed back to the calling procedure in this same manner.

6.4.4 Saving Procedure State Information

The processor does not save the contents of the general-purpose registers, segment registers, or the EFLAGS register on a procedure call. A calling procedure should explicitly save the values in any of the general-purpose registers that it will need when it resumes execution after a return. These values can be saved on the stack or in memory in one of the data segments.

The PUSHA and POPA instructions facilitate saving and restoring the contents of the general-purpose registers. PUSHA pushes the values in all the general-purpose registers on the stack in the following order: EAX, ECX, EDX, EBX, ESP (the value prior to executing the PUSHA instruction), EBP, ESI, and EDI. The POPA instruction pops all the register values saved with a PUSHA instruction (except the ESP value) from the stack to their respective registers.

If a called procedure changes the state of any of the segment registers explicitly, it should restore them to their former values before executing a return to the calling procedure.

If a calling procedure needs to maintain the state of the EFLAGS register, it can save and restore all or part of the register using the PUSHF/PUSHFD and POPF/POPFD instructions. The PUSHF instruction pushes the lower word of the EFLAGS register on the stack, while the PUSHFD instruction pushes the entire register. The POPF instruction pops a word from the stack into the lower word of the EFLAGS register, while the POPFD instruction pops a double word from the stack into the register.

6.4.5 Calls to Other Privilege Levels

The IA-32 architecture's protection mechanism recognizes four privilege levels, numbered from 0 to 3, where a greater number mean less privilege. The reason to use privilege levels is to improve the reliability of operating systems. For example, Figure 6-4 shows how privilege levels can be interpreted as rings of protection.

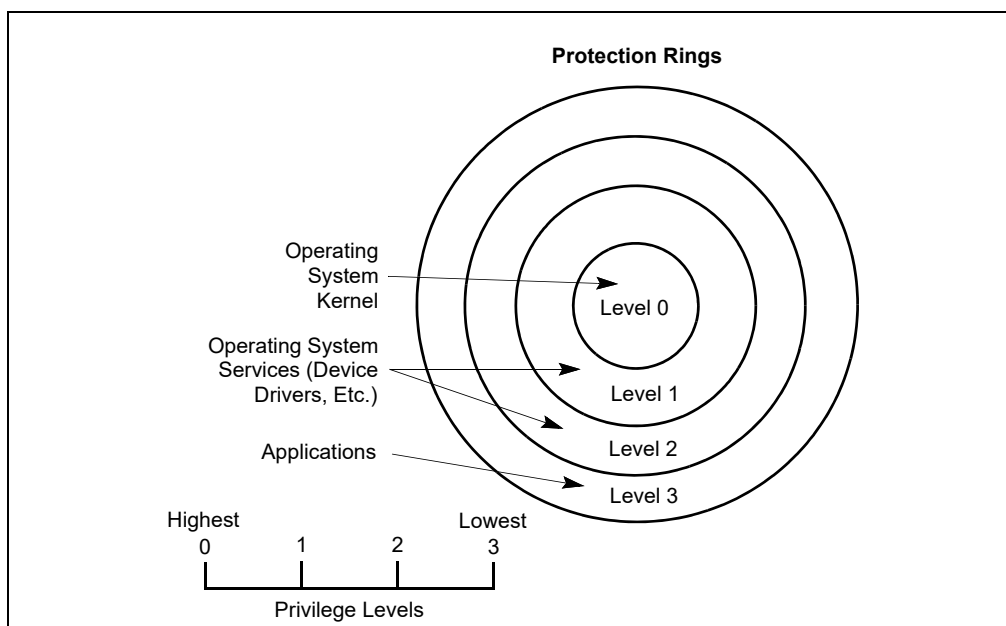


Figure 6-4. Protection Rings

In this example, the highest privilege level 0 (at the center of the diagram) is used for segments that contain the most critical code modules in the system, usually the kernel of an operating system. The outer rings (with progressively lower privileges) are used for segments that contain code modules for less critical software.

Code modules in lower privilege segments can only access modules operating at higher privilege segments by means of a tightly controlled and protected interface called a **gate**. Attempts to access higher privilege segments without going through a protection gate and without having sufficient access rights causes a general-protection exception (#GP) to be generated.

If an operating system or executive uses this multilevel protection mechanism, a call to a procedure that is in a more privileged protection level than the calling procedure is handled in a similar manner as a far call (see Section 6.4.2, "Far CALL and RET Operation"). The differences are as follows:

- The segment selector provided in the CALL instruction references a special data structure called a **call gate descriptor**. Among other things, the call gate descriptor provides the following:
 - Access rights information.
 - The segment selector for the code segment of the called procedure.
 - An offset into the code segment (that is, the instruction pointer for the called procedure).
- The processor switches to a new stack to execute the called procedure. Each privilege level has its own stack. The segment selector and stack pointer for the privilege level 3 stack are stored in the SS and ESP registers, respectively, and are automatically saved when a call to a more privileged level occurs. The segment selectors and stack pointers for the privilege level 2, 1, and 0 stacks are stored in a system segment called the task state segment (TSS).

The use of a call gate and the TSS during a stack switch are transparent to the calling procedure, except when a general-protection exception is raised.

Flexible return and event delivery (FRED) defines new mechanisms for changing privilege levels. See Chapter 8, "Flexible Return and Event Delivery (FRED)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3. When FRED transitions are enabled, call gates cannot be used to change privilege level. An execution of CALL causes a general-protection exception (#GP) if it accesses a call gate. Similarly, an execution of RET causes a #GP if it would change privilege level. In any of these cases, the fault identifies the relevant selector in its error code.

6.4.6 CALL and RET Operation Between Privilege Levels

When making a call to a more privileged protection level, the processor does the following (see Figure 6-5):

1. Performs an access rights check (privilege check).
2. Temporarily saves (internally) the current contents of the SS, ESP, CS, and EIP registers.

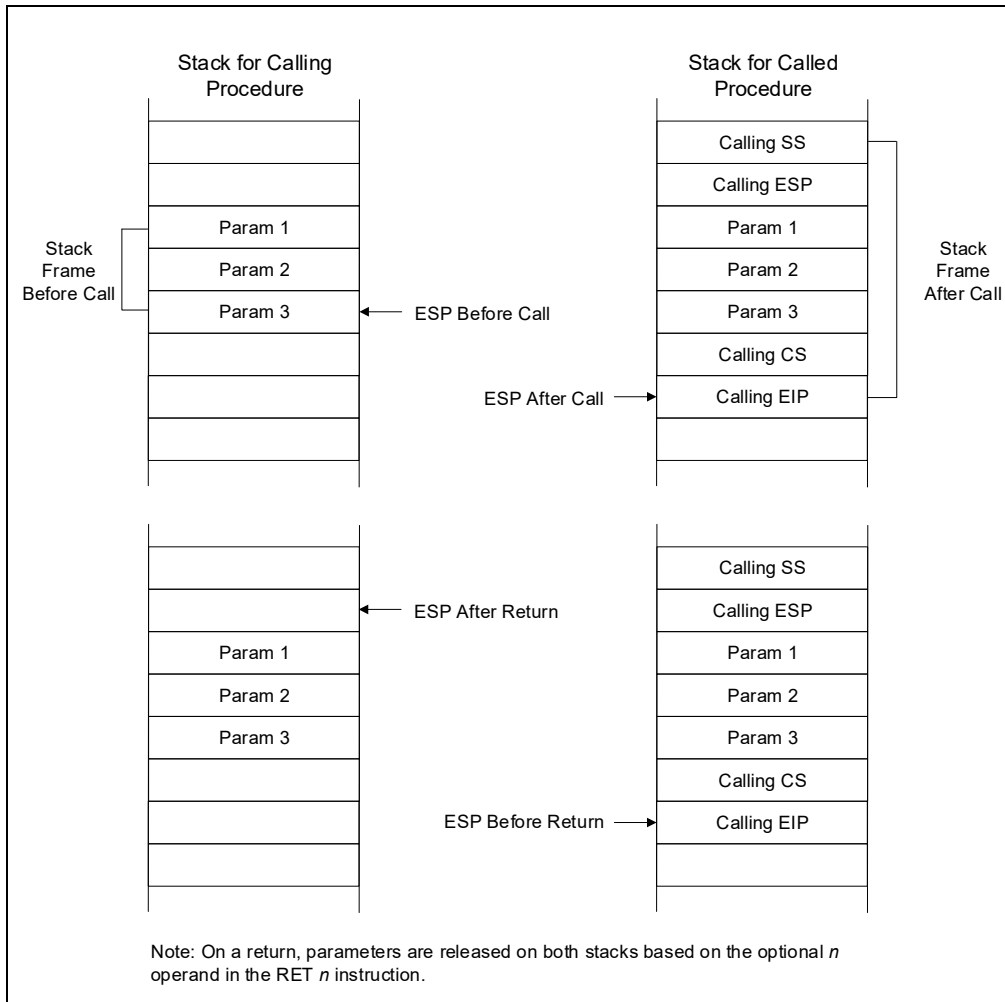


Figure 6-5. Stack Switch on a Call to a Different Privilege Level

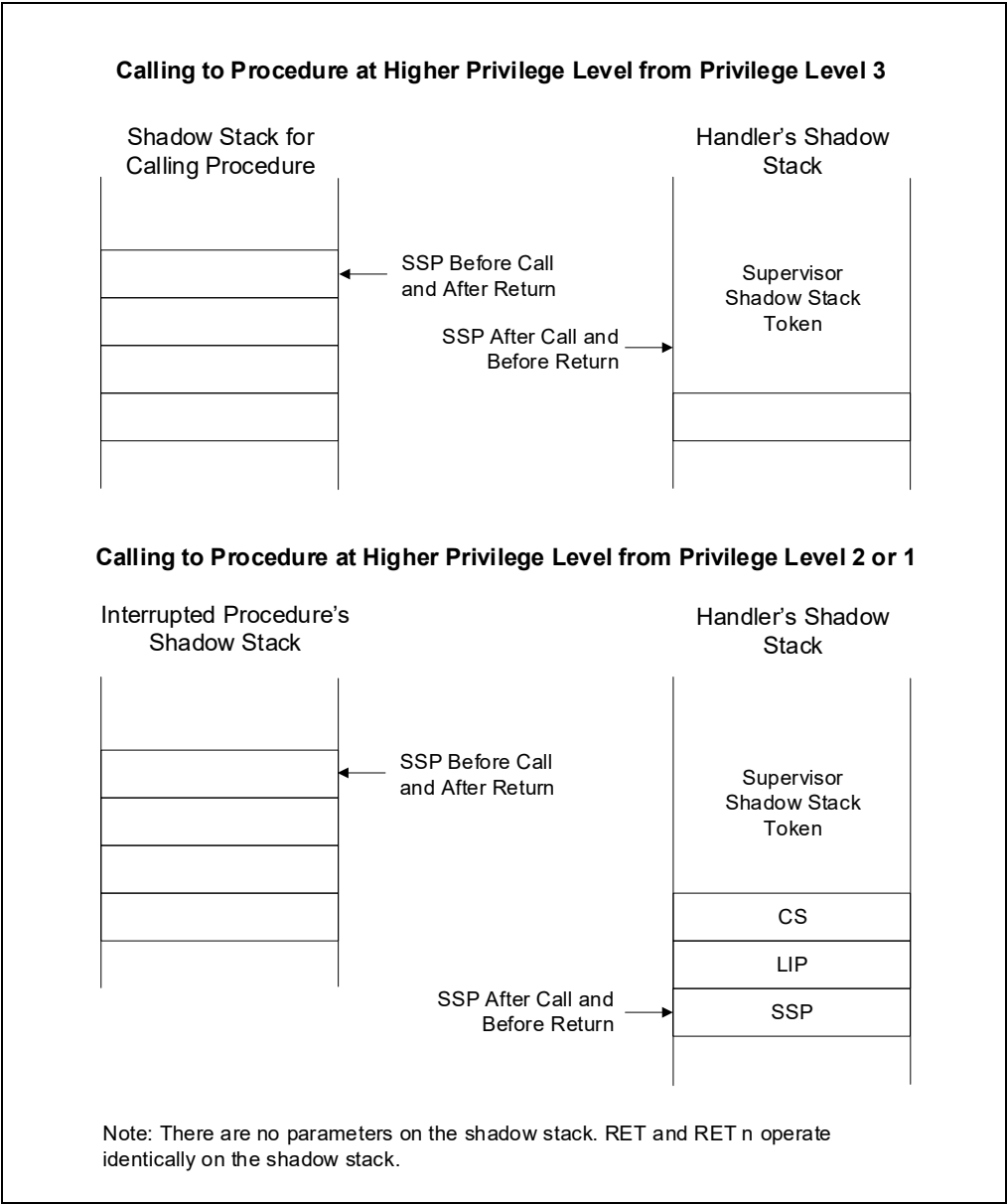


Figure 6-6. Shadow Stack Switch on a Call to a Different Privilege Level

3. Loads the segment selector and stack pointer for the new stack (that is, the stack for the privilege level being called) from the TSS into the SS and ESP registers and switches to the new stack.
 4. Pushes the temporarily saved SS and ESP values for the calling procedure's stack onto the new stack.
 5. Copies the parameters from the calling procedure's stack to the new stack. A value in the call gate descriptor determines how many parameters to copy to the new stack.
 6. Pushes the temporarily saved CS and EIP values for the calling procedure to the new stack.
- If shadow stack is enabled at the privilege level of the calling procedure, then the processor temporarily saves the SSP of the calling procedure internally. If the calling procedure is at privilege level 3, the SSP of the calling procedure is also saved into the IA32_PL3_SSP MSR.

If shadow stack is enabled at the privilege level of the called procedure, then the SSP for the called procedure is obtained from one of the MSRs listed below, depending on the target privilege level. The SSP obtained is then verified to ensure it points to a valid supervisor shadow stack that is not currently active by verifying a supervisor shadow stack token at the address pointed to by the SSP. The operations performed to verify and acquire the supervisor shadow stack token by making it busy are as described in Section 18.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

- IA32_PL2_SSP if transitioning to ring 2.
- IA32_PL1_SSP if transitioning to ring 1.
- IA32_PL0_SSP if transitioning to ring 0.

If shadow stack is enabled at the privilege level of the called procedure and the calling procedure was not at privilege level 3, then the processor pushes the temporarily saved CS, LIP (CS.base + EIP), and SSP of the calling procedure to the new shadow stack.¹

7. Loads the segment selector for the new code segment and the new instruction pointer from the call gate into the CS and EIP registers, respectively.
8. Begins execution of the called procedure at the new privilege level.

When executing a return from the privileged procedure, the processor performs these actions:

1. Performs a privilege check.
2. Restores the CS and EIP registers to their values prior to the call.

If shadow stack is enabled at the current privilege level:

- Causes a control protection exception (#CP(FAR-RET/IRET)) if SSP is not aligned to 8 bytes.
 - If the privilege level of the procedure being returned to is less than 3 (returning to supervisor mode):
 - Compares the values on shadow stack at address SSP+8 (the LIP) and SSP+16 (the CS) to the CS and (CS.base + EIP) popped from the stack and causes a control protection exception (#CP(FAR-RET/IRET)) if they do not match.
 - Temporarily saves the top-of-stack value (the SSP of the procedure being returned to) internally.
 - If a busy supervisor shadow stack token is present at address SSP+24, then marks the token free using operations described in Section 18.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.
 - If the privilege level of the procedure being returned to is less than 3 (returning to supervisor mode), restores the SSP register from the internally saved value.
 - If the privilege level of the procedure being returned to is 3 (returning to user mode) and shadow stack is enabled at privilege level 3, then restores the SSP register with value of IA32_PL3_SSP MSR.
3. If the RET instruction has an optional *n* argument, increments the stack pointer by the number of bytes specified with the *n* operand to release parameters from the stack. If the call gate descriptor specifies that one or more parameters be copied from one stack to the other, a RET *n* instruction must be used to release the parameters from both stacks. Here, the *n* operand specifies the number of bytes occupied on each stack by the parameters. On a return, the processor increments ESP by *n* for each stack to step over (effectively remove) these parameters from the stacks.
 4. Restores the SS and ESP registers to their values prior to the call, which causes a switch back to the stack of the calling procedure.
 5. If the RET instruction has an optional *n* argument, increments the stack pointer by the number of bytes specified with the *n* operand to release parameters from the stack (see explanation in step 3).
 6. Resumes execution of the calling procedure.

See Chapter 6, "Protection," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for detailed information on calls to privileged levels and the call gate descriptor.

1. If any of these pushes leads to an exception or a VM exit, the supervisor shadow-stack token remains busy.

6.4.7 Branch Functions in 64-Bit Mode

The 64-bit extensions expand branching mechanisms to accommodate branches in 64-bit linear-address space. These are:

- Near-branch semantics are redefined in 64-bit mode.
- In 64-bit mode and compatibility mode, 64-bit call-gate descriptors for far calls are available.

In 64-bit mode, the operand size for all near branches (CALL, RET, JCC, JCXZ, JMP, and LOOP) is forced to 64 bits. These instructions update the 64-bit RIP without the need for a REX operand-size prefix.

The following aspects of near branches are controlled by the effective operand size:

- Truncation of the size of the instruction pointer.
- Size of a stack pop or push, due to a CALL or RET.
- Size of a stack-pointer increment or decrement, due to a CALL or RET.
- Indirect-branch operand size.

In 64-bit mode, all of the above actions are forced to 64 bits regardless of operand size prefixes (operand size prefixes are silently ignored). However, the displacement field for relative branches is still limited to 32 bits and the address size for near branches is not forced in 64-bit mode.

Address sizes affect the size of RCX used for JCXZ and LOOP; they also impact the address calculation for memory indirect branches. Such addresses are 64 bits by default; but they can be overridden to 32 bits by an address size prefix.

Software typically uses far branches to change privilege levels. The legacy IA-32 architecture provides the call-gate mechanism to allow software to branch from one privilege level to another, although call gates can also be used for branches that do not change privilege levels. When call gates are used, the selector portion of the direct or indirect pointer references a gate descriptor (the offset in the instruction is ignored). The offset to the destination's code segment is taken from the call-gate descriptor.

64-bit mode redefines the type value of a 32-bit call-gate descriptor type to a 64-bit call gate descriptor and expands the size of the 64-bit descriptor to hold a 64-bit offset. The 64-bit mode call-gate descriptor allows far branches that reference any location in the supported linear-address space. These call gates also hold the target code selector (CS), allowing changes to privilege level and default size as a result of the gate transition.

Because immediates are generally specified up to 32 bits, the only way to specify a full 64-bit absolute RIP in 64-bit mode is with an indirect branch. For this reason, direct far branches are eliminated from the instruction set in 64-bit mode.

64-bit mode also expands the semantics of the SYSENTER and SYSEXIT instructions so that the instructions operate within a 64-bit memory space. The mode also introduces two new instructions: SYSCALL and SYSRET (which are valid only in 64-bit mode). For details, see "SYSENTER—Fast System Call," "SYSEXIT—Fast Return from Fast System Call," "SYSCALL—Fast System Call," and "SYSRET—Return From Fast System Call" in Chapter 4, "Instruction Set Reference, M-U," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B.

6.5 INTERRUPTS AND EXCEPTIONS

The processor provides two mechanisms for interrupting program execution, interrupts, and exceptions:

- An **interrupt** is an asynchronous event that is typically triggered by an I/O device.
- An **exception** is a synchronous event that is generated when the processor detects one or more predefined conditions while executing an instruction. The IA-32 architecture specifies three classes of exceptions: faults, traps, and aborts.

The processor responds to interrupts and exceptions in essentially the same way. When an interrupt or exception is signaled, the processor halts execution of the current program or task and switches to a handler procedure that has been written specifically to handle the interrupt or exception condition. The processor accesses the handler procedure through an entry in the interrupt descriptor table (IDT) [or through FRED event delivery](#). When the handler has completed handling the interrupt or exception, program control is returned to the interrupted program or task.

The operating system, executive, and/or device drivers normally handle interrupts and exceptions independently from application programs or tasks. Application programs can, however, access the interrupt and exception handlers incorporated in an operating system or executive through assembly-language calls. The remainder of this section gives a brief overview of the processor's interrupt and exception handling mechanism. See Chapter 7, "Interrupt and Exception Handling," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for a description of this mechanism.

The IA-32 Architecture defines 18 predefined interrupts and exceptions and 224 user defined interrupts, which are associated with entries in the IDT. Each interrupt and exception in the IDT is identified with a number, called a **vector**. Table 6-1 lists the interrupts and exceptions with entries in the IDT and their respective vectors. Vectors 0 through 8, 10 through 14, and 16 through 19 are the predefined interrupts and exceptions; vectors 32 through 255 are for software-defined interrupts, which are for either **software interrupts** or **maskable hardware interrupts**.

When FRED transitions are enabled, the processor does not use the IDT. Instead, interrupts and exceptions are delivered using **FRED event delivery**. FRED event delivery saves an event's vector on the stack, along with the event type (e.g., interrupt or exception). The event type allows software to distinguish interrupts and exceptions with the same vector.

Note that the processor defines several additional interrupts that do not point to entries in the IDT and are not delivered using FRED event delivery; the most notable of these interrupts is the SMI interrupt. See Chapter 7, "Interrupt and Exception Handling," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for more information about the interrupts and exceptions.

When the processor detects an interrupt or exception, it does one of the following things:

- Executes an implicit call to a handler procedure.
- Executes an implicit call to a handler task.

6.5.1 Call and Return Operation for Interrupt or Exception Handling Procedures

This section describes **IDT event delivery**, a method for calling OS handlers for interrupts and exceptions. When FRED transitions are enabled, FRED event delivery is used instead. FRED event delivery is detailed in Section 8.3, "FRED Event Delivery" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3.

A call to an interrupt or exception handler procedure is similar to a procedure call to another protection level (see Section 6.4.6, "CALL and RET Operation Between Privilege Levels"). Here, the vector references one of two kinds of gates in the IDT: an **interrupt gate** or a **trap gate**. Interrupt and trap gates are similar to call gates in that they provide the following information:

- Access rights information
- The segment selector for the code segment that contains the handler procedure
- An offset into the code segment to the first instruction of the handler procedure

The difference between an interrupt gate and a trap gate is as follows. If an interrupt or exception handler is called through an interrupt gate, the processor clears the interrupt enable (IF) flag in the EFLAGS register to prevent subsequent interrupts from interfering with the execution of the handler. When a handler is called through a trap gate, the state of the IF flag is not changed.

Table 6-1. Exceptions and Interrupts

Vector	Mnemonic	Description	Source
0	#DE	Divide Error	DIV and IDIV instructions.
1	#DB	Debug	Any code or data reference.
2		NMI Interrupt	Non-maskable external interrupt.
3	#BP	Breakpoint	INT3 instruction.
4	#OF	Overflow	INTO instruction.
5	#BR	BOUND Range Exceeded	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	UD instruction or reserved opcode.

Table 6-1. Exceptions and Interrupts (Contd.)

Vector	Mnemonic	Description	Source
7	#NM	Device Not Available (No Math Coprocessor)	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Any instruction that can generate an exception, an NMI, or an INTR.
9	#MF	CoProcessor Segment Overrun (reserved)	Floating-point instruction. ¹
10	#TS	Invalid TSS	Task switch or TSS access.
11	#NP	Segment Not Present	Loading segment registers or accessing system segments.
12	#SS	Stack Segment Fault	Stack operations and SS register loads.
13	#GP	General Protection	Any memory reference and other protection checks.
14	#PF	Page Fault	Any memory reference.
15		Reserved	
16	#MF	Floating-Point Error (Math Fault)	Floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Any data reference in memory. ²
18	#MC	Machine Check	Error codes (if any) and source are model dependent. ³
19	#XM	SIMD Floating-Point Exception	SIMD Floating-Point Instruction ⁴
20	#VE	Virtualization Exception	EPT violations ⁵
21	#CP	Control Protection Exception	The RET, IRET, RSTORSSP, and SETSSBSY instructions can generate this exception. When CET indirect branch tracking is enabled, this exception can be generated due to a missing ENDBRANCH instruction at the target of an indirect call or jump.
22-31		Reserved	
32-255		Maskable Interrupts	External interrupt from INTR pin or INT <i>n</i> instruction.

NOTES:

1. IA-32 processors after the Intel386 processor do not generate this exception.
2. This exception was introduced in the Intel486 processor.
3. This exception was introduced in the Pentium processor and enhanced in the P6 family processors.
4. This exception was introduced in the Pentium III processor.
5. This exception can occur only on processors that support the 1-setting of the "EPT-violation #VE" VM-execution control.

If the code segment for the handler procedure has the same privilege level as the currently executing program or task, the handler procedure uses the current stack; if the handler executes at a more privileged level, the processor switches to the stack for the handler's privilege level.

If no stack switch occurs, the processor does the following when calling an interrupt or exception handler (see Figure 6-7):

1. Pushes the current contents of the EFLAGS, CS, and EIP registers (in that order) on the stack.
If shadow stack is enabled:
 - a. Temporarily saves the current value of the SSP register internally.
 - b. Pushes the current value of the CS register on the shadow stack.
 - c. Pushes the current value of LIP (CS.base + EIP) on the shadow stack.
 - d. Pushes the temporarily saved SSP value on the shadow stack.
2. Pushes an error code (if appropriate) on the stack.
3. Loads the segment selector for the new code segment and the new instruction pointer (from the interrupt gate or trap gate) into the CS and EIP registers, respectively.
4. If the call is through an interrupt gate, clears the IF flag in the EFLAGS register.

5. Begins execution of the handler procedure.

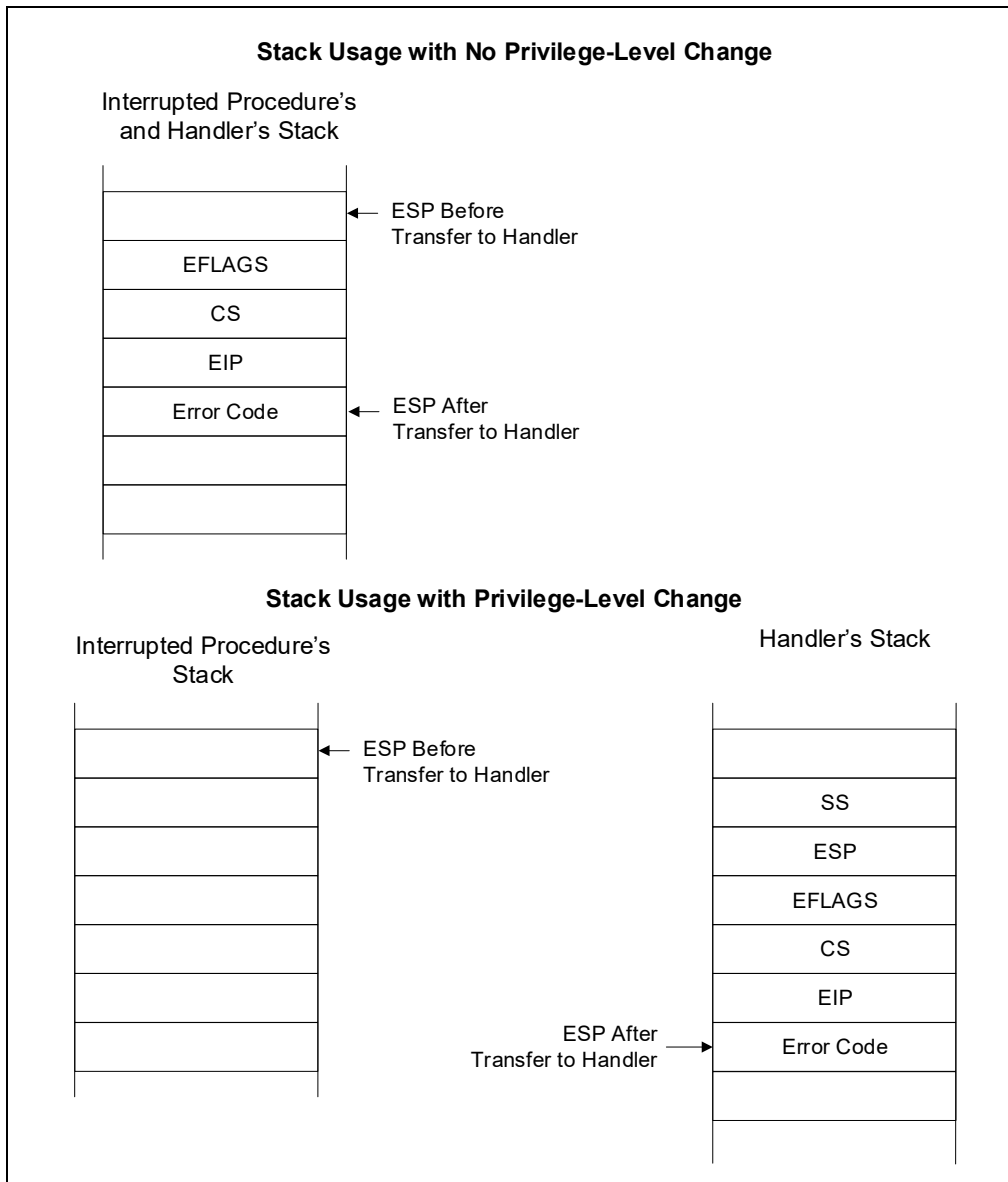


Figure 6-7. Stack Usage on Transfers to Interrupt and Exception Handling Routines

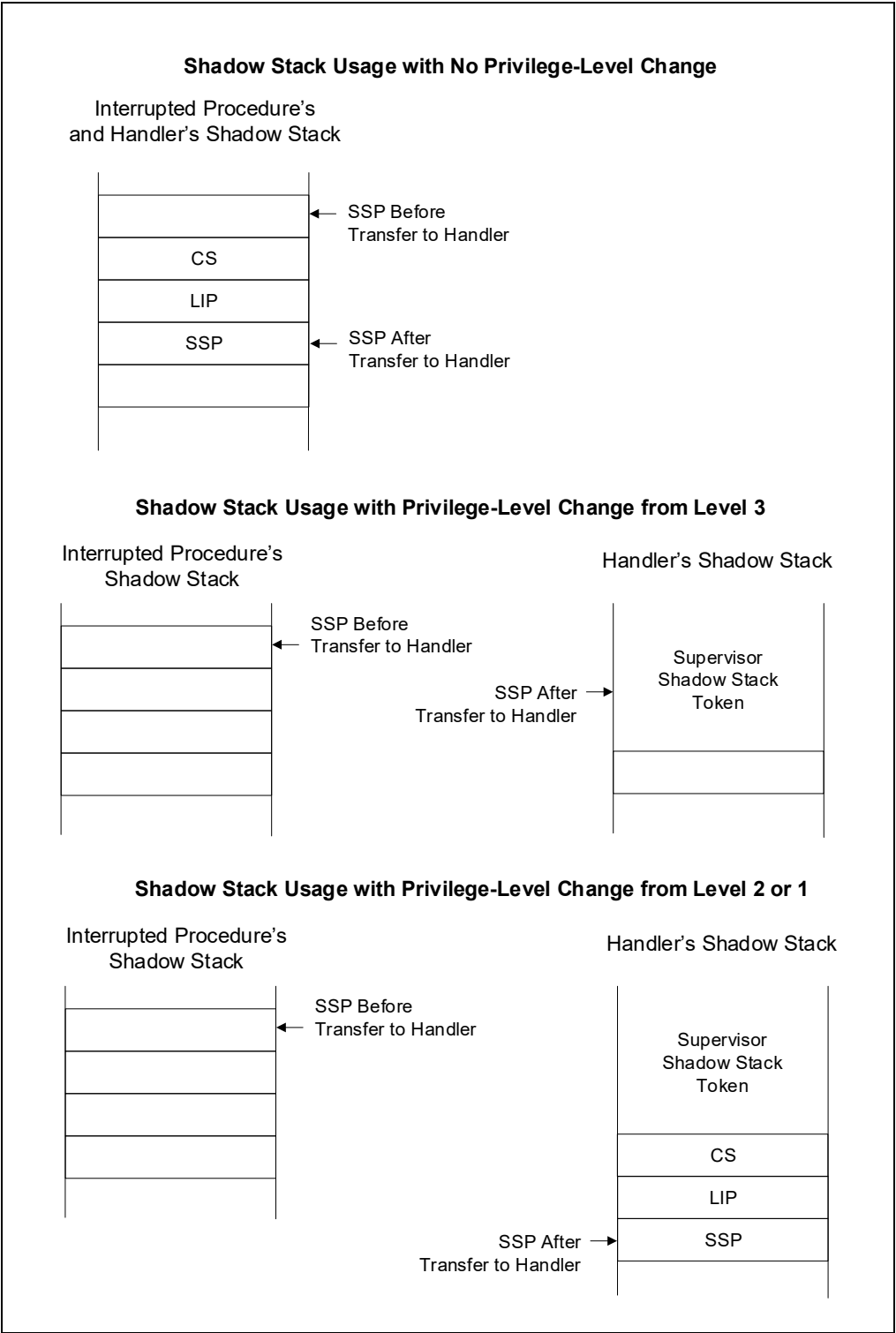


Figure 6-8. Shadow Stack Usage on Transfers to Interrupt and Exception Handling Routines

If a stack switch does occur, the processor does the following:

1. Temporarily saves (internally) the current contents of the SS, ESP, EFLAGS, CS, and EIP registers.
2. Loads the segment selector and stack pointer for the new stack (that is, the stack for the privilege level being called) from the TSS into the SS and ESP registers and switches to the new stack.
3. Pushes the temporarily saved SS, ESP, EFLAGS, CS, and EIP values for the interrupted procedure's stack onto the new stack.

If shadow stack is enabled at the privilege level of the interrupted procedure, then the processor temporarily saves the SSP of the interrupted procedure internally. If the interrupted procedure is at privilege level 3, the SSP of the interrupted procedure is also saved into the IA32_PL3_SSP MSR.

If shadow stack is enabled at the privilege level being called, then the SSP for the called privilege level is obtained from one of the MSRs listed below, depending on the target privilege level. The SSP obtained is then verified to ensure it points to a valid supervisor shadow stack that is not currently active by verifying a supervisor shadow stack token at the address pointed to by the SSP. The operations performed to verify and acquire the supervisor shadow stack token by making it busy are as described in Section 18.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

- IA32_PL2_SSP if transitioning to ring 2.
- IA32_PL1_SSP if transitioning to ring 1.
- IA32_PL0_SSP if transitioning to ring 0.

If shadow stack is enabled at the privilege level being called and the interrupted procedure was not at privilege level 3, then the processor pushes the temporarily saved CS, LIP (CS.base + EIP), and SSP of the interrupted procedure to the new shadow stack.¹

4. Pushes an error code on the new stack (if appropriate).
5. Loads the segment selector for the new code segment and the new instruction pointer (from the interrupt gate or trap gate) into the CS and EIP registers, respectively.
6. If the call is through an interrupt gate, clears the IF flag in the EFLAGS register.
7. Begins execution of the handler procedure at the new privilege level.

A return from an interrupt or exception handler is initiated with the IRET instruction. The IRET instruction is similar to the far RET instruction, except that it also restores the contents of the EFLAGS register for the interrupted procedure. When executing a return from an interrupt or exception handler from the same privilege level as the interrupted procedure, the processor performs these actions:

1. Restores the CS and EIP registers to their values prior to the interrupt or exception.

If shadow stack is enabled:

- a. Compares the values on the shadow stack at address SSP+8 (the LIP) and SSP+16 (the CS) to the CS and (CS.base + EIP) popped from the stack, and causes a control protection exception (#CP(FAR-RET/IRET)) if they do not match.
- b. Pops the top-of-stack value (the SSP prior to the interrupt or exception) from the shadow stack into the SSP register.

2. Restores the EFLAGS register.
3. Increments the stack pointer appropriately.
4. Resumes execution of the interrupted procedure.

When executing a return from an interrupt or exception handler from a different privilege level than the interrupted procedure, the processor performs these actions:

1. Performs a privilege check.
2. Restores the CS and EIP registers to their values prior to the interrupt or exception.
3. Restores the EFLAGS register.

1. If any of these pushes leads to an exception or a VM exit, the supervisor shadow-stack token remains busy.

If shadow stack is enabled at the current privilege level:

- If SSP is not aligned to 8 bytes, then causes a control protection exception (`#CP(FAR-RET/IRET)`).
 - If the privilege level of the procedure being returned to is less than 3 (returning to supervisor mode):
 - Compares the values on the shadow stack at address `SSP+8` (the LIP) and `SSP+16` (the CS) to the CS and `(CS.base + EIP)` popped from the stack, and causes a control protection exception (`#CP(FAR-RET/IRET)`) if they do not match.
 - Temporarily saves the top-of-stack value (the SSP of the procedure being returned to) internally.
 - If a busy supervisor shadow stack token is present at address `SSP+24`, then marks the token free using operations described in Section 18.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.
 - If the privilege level of the procedure being returned to is less than 3 (returning to supervisor mode), restores the SSP register from the internally saved value.
 - If the privilege level of the procedure being returned to is 3 (returning to user mode) and shadow stack is enabled at privilege level 3, then restores the SSP register with the value of the `IA32_PL3_SSP` MSR.
4. Restores the SS and ESP registers to their values prior to the interrupt or exception, resulting in a stack switch back to the stack of the interrupted procedure.
 5. Resumes execution of the interrupted procedure.

6.5.2 Calls to Interrupt or Exception Handler Tasks

Interrupt and exception handler routines can also be executed in a separate task. Here, an interrupt or exception causes a task switch to a handler task. The handler task is given its own address space and (optionally) can execute at a higher protection level than application programs or tasks.

The switch to the handler task is accomplished with an implicit task call that references a **task gate descriptor**. The task gate provides access to the address space for the handler task. As part of the task switch, the processor saves complete state information for the interrupted program or task. Upon returning from the handler task, the state of the interrupted program or task is restored and execution continues. See Chapter 7, "Interrupt and Exception Handling," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for more information on handling interrupts and exceptions through handler tasks.

6.5.3 Interrupt and Exception Handling in Real-Address Mode

When operating in real-address mode, the processor responds to an interrupt or exception with an implicit far call to an interrupt or exception handler. The processor uses the interrupt or exception vector as an index into an interrupt table. The interrupt table contains instruction pointers to the interrupt and exception handler procedures.

The processor saves the state of the EFLAGS register, the EIP register, the CS register, and an optional error code on the stack before switching to the handler procedure.

A return from the interrupt or exception handler is carried out with the IRET instruction.

See Chapter 23, "8086 Emulation," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, for more information on handling interrupts and exceptions in real-address mode.

6.5.4 INT *n*, INTO, INT3, INT1, and BOUND Instructions

The `INT n`, `INTO`, `INT3`, and `BOUND` instructions allow a program or task to explicitly call an interrupt or exception handler. The `INT n` instruction (opcode `CD`) uses a vector as an argument, which allows a program to call any interrupt handler.

The `INTO` instruction (opcode `CE`) explicitly calls the overflow exception (`#OF`) handler if the overflow flag (`OF`) in the EFLAGS register is set. The `OF` flag indicates overflow on arithmetic instructions, but it does not automatically raise an overflow exception. An overflow exception can only be raised explicitly in either of the following ways:

- Execute the INTO instruction.
- Test the OF flag and execute the INT *n* instruction with an argument of 4 (the vector of the overflow exception) if the flag is set.

Both the methods of dealing with overflow conditions allow a program to test for overflow at specific places in the instruction stream.

The INT3 instruction (opcode CC) explicitly calls the breakpoint exception (#BP) handler. Similarly, the INT1 instruction (opcode F1) explicitly calls the debug exception (#DB) handler.¹

The BOUND instruction explicitly calls the BOUND-range exceeded exception (#BR) handler if an operand is found to be not within predefined boundaries in memory. This instruction is provided for checking references to arrays and other data structures. Like the overflow exception, the BOUND-range exceeded exception can only be raised explicitly with the BOUND instruction or the INT *n* instruction with an argument of 5 (the vector of the bounds-check exception). The processor does not implicitly perform bounds checks and raise the BOUND-range exceeded exception.

6.5.5 Handling Floating-Point Exceptions

When operating on individual or packed floating-point values, the IA-32 architecture supports a set of six floating-point exceptions. These exceptions can be generated during operations performed by the x87 FPU instructions or by SSE/SSE2/SSE3 instructions. When an x87 FPU instruction (including the FISTTP instruction in SSE3) generates one or more of these exceptions, it in turn generates floating-point error exception (#MF); when an SSE/SSE2/SSE3 instruction generates a floating-point exception, it in turn generates SIMD floating-point exception (#XM).

See the following sections for further descriptions of the floating-point exceptions, how they are generated, and how they are handled:

- Section 4.9.1, “Floating-Point Exception Conditions,” and Section 4.9.3, “Typical Actions of a Floating-Point Exception Handler.”
- Section 8.4, “x87 FPU Floating-Point Exception Handling,” and Section 8.5, “x87 FPU Floating-Point Exception Conditions.”
- Section 11.5.1, “SIMD Floating-Point Exceptions.”
- Interrupt Behavior.

6.5.6 Interrupt and Exception Behavior in 64-Bit Mode

64-bit extensions expand the legacy IA-32 interrupt-processing and exception-processing mechanism to allow support for 64-bit operating systems and applications. Changes include:

- All interrupt handlers pointed to by the IDT are 64-bit code (does not apply to the SMI handler).
- The size of interrupt-stack pushes is fixed at 64 bits. The processor uses 8-byte, zero extended stores.
- The stack pointer (SS:RSP) is pushed unconditionally on interrupts. In legacy environments, this push is conditional and based on a change in current privilege level (CPL).
- The new SS is set to NULL if there is a change in CPL.
- IRET behavior changes.
- There is a new interrupt stack-switch mechanism and a new interrupt shadow stack-switch mechanism.
- The alignment of interrupt stack frame is different.

The above items apply to IDT event delivery. When FRED transitions are enabled, FRED event delivery is used instead. See Section 8.3, “FRED Event Delivery” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3.

1. Hardware vendors may use the INT1 instruction for hardware debug. For that reason, Intel recommends software vendors instead use the INT3 instruction for software breakpoints.

6.6 PROCEDURE CALLS FOR BLOCK-STRUCTURED LANGUAGES

The IA-32 architecture supports an alternate method of performing procedure calls with the ENTER (enter procedure) and LEAVE (leave procedure) instructions. These instructions automatically create and release, respectively, stack frames for called procedures. The stack frames have predefined spaces for local variables and the necessary pointers to allow coherent returns from called procedures. They also allow scope rules to be implemented so that procedures can access their own local variables and some number of other variables located in other stack frames.

ENTER and LEAVE offer two benefits:

- They provide machine-language support for implementing block-structured languages, such as C and Pascal.
- They simplify procedure entry and exit in compiler-generated code.

6.6.1 ENTER Instruction

The ENTER instruction creates a stack frame compatible with the scope rules typically used in block-structured languages. In block-structured languages, the scope of a procedure is the set of variables to which it has access. The rules for scope vary among languages. They may be based on the nesting of procedures, the division of the program into separately compiled files, or some other modularization scheme.

ENTER has two operands. The first specifies the number of bytes to be reserved on the stack for dynamic storage for the procedure being called. Dynamic storage is the memory allocated for variables created when the procedure is called, also known as automatic variables. The second parameter is the lexical nesting level (from 0 to 31) of the procedure. The nesting level is the depth of a procedure in a hierarchy of procedure calls. The lexical level is unrelated to either the protection privilege level or to the I/O privilege level of the currently running program or task.

ENTER, in the following example, allocates 2 Kbytes of dynamic storage on the stack and sets up pointers to two previous stack frames in the stack frame for this procedure:

```
ENTER 2048,3
```

The lexical nesting level determines the number of stack frame pointers to copy into the new stack frame from the preceding frame. A stack frame pointer is a doubleword used to access the variables of a procedure. The set of stack frame pointers used by a procedure to access the variables of other procedures is called the display. The first doubleword in the display is a pointer to the previous stack frame. This pointer is used by a LEAVE instruction to undo the effect of an ENTER instruction by discarding the current stack frame.

After the ENTER instruction creates the display for a procedure, it allocates the dynamic local variables for the procedure by decrementing the contents of the ESP register by the number of bytes specified in the first parameter. This new value in the ESP register serves as the initial top-of-stack for all PUSH and POP operations within the procedure.

To allow a procedure to address its display, the ENTER instruction leaves the EBP register pointing to the first doubleword in the display. Because stacks grow down, this is actually the doubleword with the highest address in the display. Data manipulation instructions that specify the EBP register as a base register automatically address locations within the stack segment instead of the data segment.

The ENTER instruction can be used in two ways: nested and non-nested. If the lexical level is 0, the non-nested form is used. The non-nested form pushes the contents of the EBP register on the stack, copies the contents of the ESP register into the EBP register, and subtracts the first operand from the contents of the ESP register to allocate dynamic storage. The non-nested form differs from the nested form in that no stack frame pointers are copied. The nested form of the ENTER instruction occurs when the second parameter (lexical level) is not zero.

The following pseudo code shows the formal definition of the ENTER instruction. STORAGE is the number of bytes of dynamic storage to allocate for local variables, and LEVEL is the lexical nesting level.


```

PUSH EBP;
FRAME_PTR := ESP;
IF LEVEL > 0
  THEN
    DO (LEVEL - 1) times
      EBP := EBP - 4;
      PUSH Pointer(EBP); (* doubleword pointed to by EBP *)
    OD;
  PUSH FRAME_PTR;
FI;
EBP := FRAME_PTR;
ESP := ESP - STORAGE;

```

The main procedure (in which all other procedures are nested) operates at the highest lexical level, level 1. The first procedure it calls operates at the next deeper lexical level, level 2. A level 2 procedure can access the variables of the main program, which are at fixed locations specified by the compiler. In the case of level 1, the ENTER instruction allocates only the requested dynamic storage on the stack because there is no previous display to copy.

A procedure that calls another procedure at a lower lexical level gives the called procedure access to the variables of the caller. The ENTER instruction provides this access by placing a pointer to the calling procedure's stack frame in the display.

A procedure that calls another procedure at the same lexical level should not give access to its variables. In this case, the ENTER instruction copies only that part of the display from the calling procedure which refers to previously nested procedures operating at higher lexical levels. The new stack frame does not include the pointer for addressing the calling procedure's stack frame.

The ENTER instruction treats a re-entrant procedure as a call to a procedure at the same lexical level. In this case, each succeeding iteration of the re-entrant procedure can address only its own variables and the variables of the procedures within which it is nested. A re-entrant procedure always can address its own variables; it does not require pointers to the stack frames of previous iterations.

By copying only the stack frame pointers of procedures at higher lexical levels, the ENTER instruction makes certain that procedures access only those variables of higher lexical levels, not those at parallel lexical levels (see Figure 6-9).

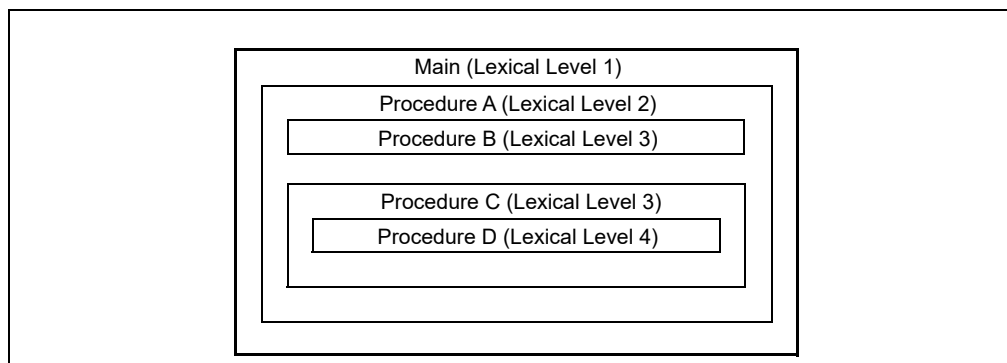


Figure 6-9. Nested Procedures

Block-structured languages can use the lexical levels defined by ENTER to control access to the variables of nested procedures. In Figure 6-9, for example, if procedure A calls procedure B which, in turn, calls procedure C, then procedure C will have access to the variables of the MAIN procedure and procedure A, but not those of procedure B because they are at the same lexical level. The following definition describes the access to variables for the nested procedures in Figure 6-9.

1. MAIN has variables at fixed locations.
2. Procedure A can access only the variables of MAIN.

- 3. Procedure B can access only the variables of procedure A and MAIN. Procedure B cannot access the variables of procedure C or procedure D.
- 4. Procedure C can access only the variables of procedure A and MAIN. Procedure C cannot access the variables of procedure B or procedure D.
- 5. Procedure D can access the variables of procedure C, procedure A, and MAIN. Procedure D cannot access the variables of procedure B.

In Figure 6-10, an ENTER instruction at the beginning of the MAIN procedure creates three doublewords of dynamic storage for MAIN, but copies no pointers from other stack frames. The first doubleword in the display holds a copy of the last value in the EBP register before the ENTER instruction was executed. The second doubleword holds a copy of the contents of the EBP register following the ENTER instruction. After the instruction is executed, the EBP register points to the first doubleword pushed on the stack, and the ESP register points to the last doubleword in the stack frame.

When MAIN calls procedure A, the ENTER instruction creates a new display (see Figure 6-11). The first doubleword is the last value held in MAIN's EBP register. The second doubleword is a pointer to MAIN's stack frame which is copied from the second doubleword in MAIN's display. This happens to be another copy of the last value held in MAIN's EBP register. Procedure A can access variables in MAIN because MAIN is at level 1.

Therefore the base address for the dynamic storage used in MAIN is the current address in the EBP register, plus four bytes to account for the saved contents of MAIN's EBP register. All dynamic variables for MAIN are at fixed, positive offsets from this value.

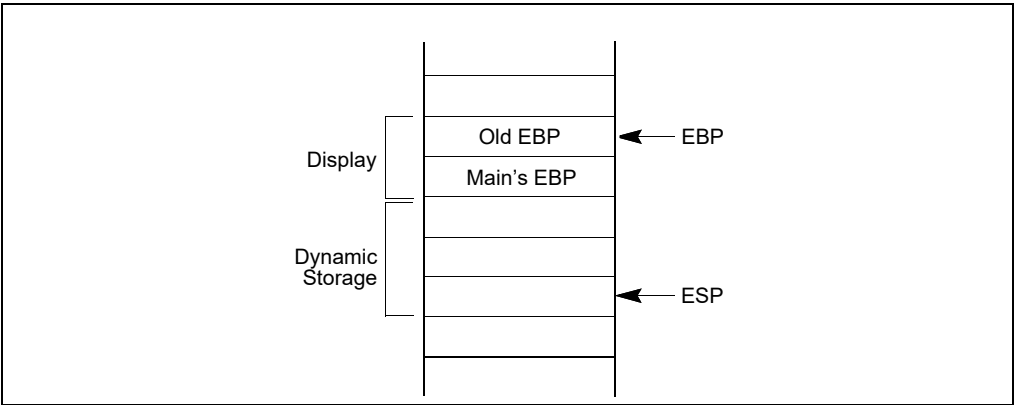


Figure 6-10. Stack Frame After Entering the MAIN Procedure

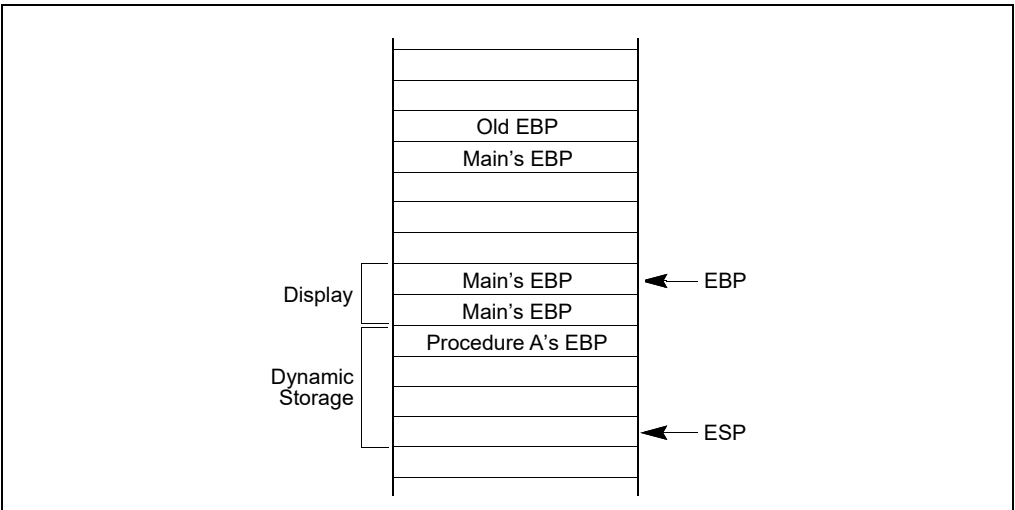


Figure 6-11. Stack Frame After Entering Procedure A

When procedure A calls procedure B, the ENTER instruction creates a new display (see Figure 6-12). The first doubleword holds a copy of the last value in procedure A's EBP register. The second and third doublewords are copies of the two stack frame pointers in procedure A's display. Procedure B can access variables in procedure A and MAIN by using the stack frame pointers in its display.

When procedure B calls procedure C, the ENTER instruction creates a new display for procedure C (see Figure 6-13). The first doubleword holds a copy of the last value in procedure B's EBP register. This is used by the LEAVE instruction to restore procedure B's stack frame. The second and third doublewords are copies of the two stack frame pointers in procedure A's display. If procedure C were at the next deeper lexical level from procedure B, a fourth doubleword would be copied, which would be the stack frame pointer to procedure B's local variables.

Note that procedure B and procedure C are at the same level, so procedure C is not intended to access procedure B's variables. This does not mean that procedure C is completely isolated from procedure B; procedure C is called by procedure B, so the pointer to the returning stack frame is a pointer to procedure B's stack frame. In addition, procedure B can pass parameters to procedure C either on the stack or through variables global to both procedures (that is, variables in the scope of both procedures).

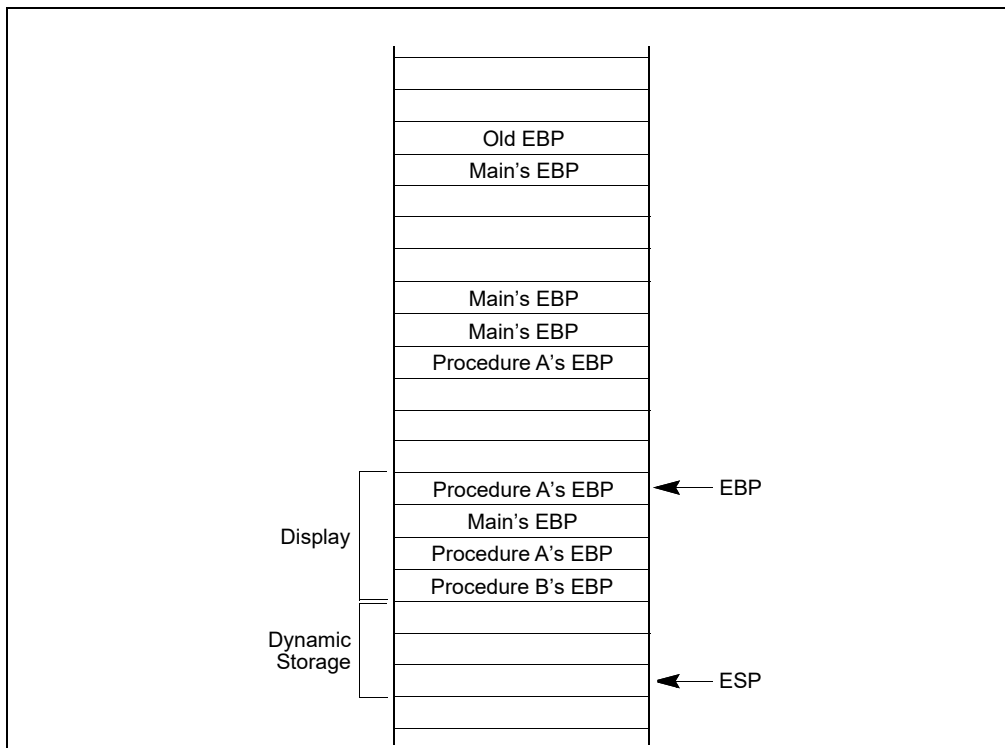


Figure 6-12. Stack Frame After Entering Procedure B

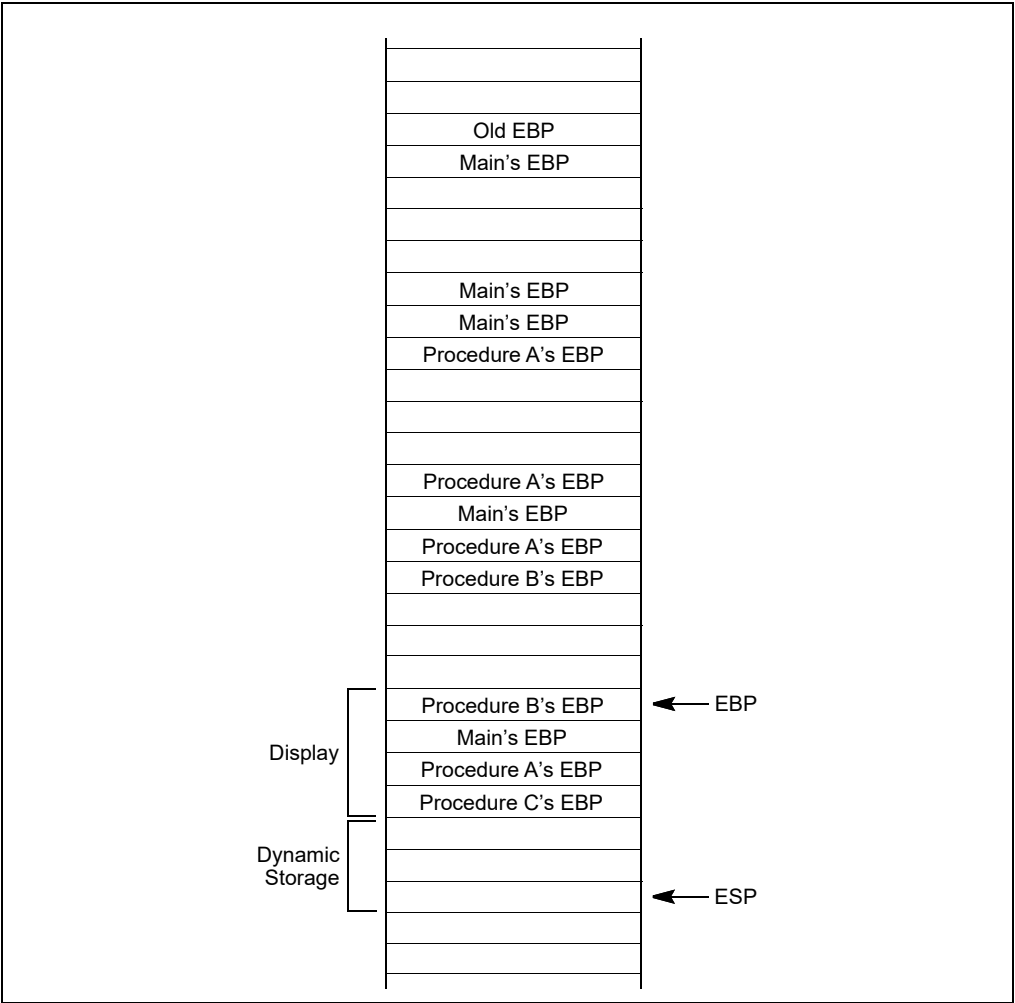


Figure 6-13. Stack Frame After Entering Procedure C

6.6.2 LEAVE Instruction

The LEAVE instruction, which does not have any operands, reverses the action of the previous ENTER instruction. The LEAVE instruction copies the contents of the EBP register into the ESP register to release all stack space allocated to the procedure. Then it restores the old value of the EBP register from the stack. This simultaneously restores the ESP register to its original value. A subsequent RET instruction then can remove any arguments and the return address pushed on the stack by the calling program for use by the procedure.

3. Updates to Chapter 7, Volume 1

Change bars and text show changes to Chapter 7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

Changes to this chapter:

- In Section 7.3.9.3, "Fast-String Operation," updated content clarifying that fast-string operations may perform duplicate writes to certain memory addresses.

CHAPTER 7

PROGRAMMING WITH GENERAL-PURPOSE INSTRUCTIONS

General-purpose (GP) instructions are a subset of the IA-32 instructions that represent the fundamental instruction set for the Intel IA-32 processors. These instructions were introduced into the IA-32 architecture with the first IA-32 processors (the Intel 8086 and 8088). Additional instructions were added to the general-purpose instruction set in subsequent families of IA-32 processors (the Intel 286, Intel386, Intel486, Pentium, Pentium Pro, and Pentium II processors).

Intel 64 architecture further extends the capability of most general-purpose instructions so that they are able to handle 64-bit data in 64-bit mode. A small number of general-purpose instructions (still supported in non-64-bit modes) are not supported in 64-bit mode.

General-purpose instructions perform basic data movement, memory addressing, arithmetic and logical, program flow control, input/output, and string operations on a set of integer, pointer, and BCD data types. This chapter provides an overview of the general-purpose instructions. See the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D, for detailed descriptions of individual instructions.

7.1 PROGRAMMING ENVIRONMENT FOR GP INSTRUCTIONS

The programming environment for the general-purpose instructions consists of the set of registers and address space. The environment includes the following items:

- **General-purpose registers** — Eight 32-bit general-purpose registers (see Section 3.4.1, "General-Purpose Registers") are used in non-64-bit modes to address operands in memory. These registers are referenced by the names EAX, EBX, ECX, EDX, EBP, ESI, EDI, and ESP.
- **Segment registers** — The six 16-bit segment registers contain segment pointers for use in accessing memory (see Section 3.4.2, "Segment Registers"). These registers are referenced by the names CS, DS, SS, ES, FS, and GS.
- **EFLAGS register** — This 32-bit register (see Section 3.4.3, "EFLAGS Register") is used to provide status and control for basic arithmetic, compare, and system operations.
- **EIP register** — This 32-bit register contains the current instruction pointer (see Section 3.5, "Instruction Pointer").

General-purpose instructions operate on the following data types. The width of valid data types is dependent on processor mode (see Chapter 4):

- Bytes, words, doublewords.
- Signed and unsigned byte, word, doubleword integers.
- Near and far pointers.
- Bit fields.
- BCD integers.

7.2 PROGRAMMING ENVIRONMENT FOR GP INSTRUCTIONS IN 64-BIT MODE

The programming environment for the general-purpose instructions in 64-bit mode is similar to that described in Section 7.1.

- **General-purpose registers** — In 64-bit mode, sixteen general-purpose registers are available. These include the eight GPRs described in Section 7.1 and eight new GPRs (R8D-R15D). R8D-R15D are available by using a REX prefix. All sixteen GPRs can be promoted to 64 bits. The 64-bit registers are referenced as RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, and R8-R15 (see Section 3.4.1.1, "General-Purpose Registers in 64-Bit Mode"). Promotion to 64-bit operand requires REX prefix encodings.

- **Segment registers** — In 64-bit mode, segmentation is available but it is set up uniquely (see Section 3.4.2.1, “Segment Registers in 64-Bit Mode”).
- **Flags and Status register** — When the processor is running in 64-bit mode, EFLAGS becomes the 64-bit RFLAGS register (see Section 3.4.3, “EFLAGS Register”).
- **Instruction Pointer register** — In 64-bit mode, the EIP register becomes the 64-bit RIP register (see Section 3.5.1, “Instruction Pointer in 64-Bit Mode”).

General-purpose instructions operate on the following data types in 64-bit mode. The width of valid data types is dependent on default operand size, address size, or a prefix that overrides the default size:

- Bytes, words, doublewords, quadwords.
- Signed and unsigned byte, word, doubleword, quadword integers.
- Near and far pointers.
- Bit fields.

See also:

- Chapter 3, “Basic Execution Environment,” for more information about IA-32e modes.
- Chapter 2, “Instruction Format,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A, for more detailed information about REX prefixes.
- Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B, 2C, & 2D, for a complete listing of all instructions. This information documents the behavior of individual instructions in the 64-bit mode context.

7.3 SUMMARY OF GP INSTRUCTIONS

General purpose instructions are divided into the following subgroups:

- Data transfer.
- Binary arithmetic.
- Decimal arithmetic.
- Logical.
- Shift and rotate.
- Bit and byte.
- Control transfer.
- String.
- I/O.
- Enter and Leave.
- Flag control.
- Segment register.
- Miscellaneous.

Each sub-group of general-purpose instructions is discussed in the context of non-64-bit mode operation first. Changes in 64-bit mode beyond those affected by the use of the REX prefixes are discussed in separate sub-sections within each subgroup. For a simple list of general-purpose instructions by subgroup, see Chapter 5.

7.3.1 Data Transfer Instructions

The data transfer instructions move bytes, words, doublewords, or quadwords both between memory and the processor’s registers and between registers. For the purpose of this discussion, these instructions are divided into subordinate subgroups that provide for:

- General data movement.

- Exchange.
- Stack manipulation.
- Type conversion.

7.3.1.1 General Data Movement Instructions

Move instructions — The MOV (move) and CMOVcc (conditional move) instructions transfer data between memory and registers or between registers.

The MOV instruction performs basic load data and store data operations between memory and the processor's registers and data movement operations between registers. It handles data transfers along the paths listed in Table 7-1. (See "MOV—Move to/from Control Registers" and "MOV—Move to/from Debug Registers" in Chapter 4, "Instruction Set Reference, M-U," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for information on moving data to and from the control and debug registers.)

The MOV instruction cannot move data from one memory location to another or from one segment register to another segment register. Memory-to-memory moves are performed with the MOVS (string move) instruction (see Section 7.3.9, "String Operations").

Conditional move instructions — The CMOVcc instructions are a group of instructions that check the state of the status flags in the EFLAGS register and perform a move operation if the flags are in a specified state. These instructions can be used to move a 16-bit or 32-bit value from memory to a general-purpose register or from one general-purpose register to another. The flag state being tested is specified with a condition code (cc) associated with the instruction. If the condition is not satisfied, a move is not performed and execution continues with the instruction following the CMOVcc instruction.

Table 7-1. Move Instruction Operations

Type of Data Movement	Source → Destination
From memory to a register	Memory location → General-purpose register Memory location → Segment register
From a register to memory	General-purpose register → Memory location Segment register → Memory location
Between registers	General-purpose register → General-purpose register General-purpose register → Segment register Segment register → General-purpose register General-purpose register → Control register Control register → General-purpose register General-purpose register → Debug register Debug register → General-purpose register
Immediate data to a register	Immediate → General-purpose register
Immediate data to memory	Immediate → Memory location

Table 7-2 shows mnemonics for CMOVcc instructions and the conditions being tested for each instruction. The condition code mnemonics are appended to the letters "CMOV" to form the mnemonics for CMOVcc instructions. The instructions listed in Table 7-2 as pairs (for example, CMOVA/CMOVNBE) are alternate names for the same instruction. The assembler provides these alternate names to make it easier to read program listings.

CMOVcc instructions are useful for optimizing small IF constructions. They also help eliminate branching overhead for IF statements and the possibility of branch mispredictions by the processor.

These conditional move instructions are supported in the P6 family, Pentium 4, and Intel Xeon processors. Software can check if CMOVcc instructions are supported by checking the processor's feature information with the CPUID instruction.

7.3.1.2 Exchange Instructions

The exchange instructions swap the contents of one or more operands and, in some cases, perform additional operations such as asserting the LOCK signal or modifying flags in the EFLAGS register.

The XCHG (exchange) instruction swaps the contents of two operands. This instruction takes the place of three MOV instructions and does not require a temporary location to save the contents of one operand location while the other is being loaded. When a memory operand is used with the XCHG instruction, the processor's LOCK signal is automatically asserted. This instruction is thus useful for implementing semaphores or similar data structures for process synchronization. See "Bus Locking" in Chapter 11, "Multiple-Processor Management," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for more information on bus locking.

The BSWAP (byte swap) instruction reverses the byte order in a 32-bit register operand. Bit positions 0 through 7 are exchanged with 24 through 31, and bit positions 8 through 15 are exchanged with 16 through 23. Executing this instruction twice in a row leaves the register with the same value as before. The BSWAP instruction is useful for converting between "big-endian" and "little-endian" data formats. This instruction also speeds execution of decimal arithmetic. (The XCHG instruction can be used to swap the bytes in a word.)

Table 7-2. Conditional Move Instructions

Instruction Mnemonic	Status Flag States	Condition Description
Unsigned Conditional Moves		
CMOVA/CMOVNBE	(CF or ZF) = 0	Above/not below or equal
CMOVAE/CMOVNB	CF = 0	Above or equal/not below
CMOVNC	CF = 0	Not carry
CMOVNB/CMOVNAE	CF = 1	Below/not above or equal
CMOVNC	CF = 1	Carry
CMOVBE/CMOVNA	(CF or ZF) = 1	Below or equal/not above
CMOVE/CMOVZ	ZF = 1	Equal/zero
CMOVNE/CMOVNZ	ZF = 0	Not equal/not zero
CMOVP/CMOVPE	PF = 1	Parity/parity even
CMOVNP/CMOVPO	PF = 0	Not parity/parity odd
Signed Conditional Moves		
CMOVGE/CMOVNL	(SF xor OF) = 0	Greater or equal/not less
CMOVL/CMOVNGE	(SF xor OF) = 1	Less/not greater or equal
CMOVLE/CMOVNG	((SF xor OF) or ZF) = 1	Less or equal/not greater
CMOVO	OF = 1	Overflow
CMOVNO	OF = 0	Not overflow
CMOVS	SF = 1	Sign (negative)
CMOVNS	SF = 0	Not sign (non-negative)

The XADD (exchange and add) instruction swaps two operands and then stores the sum of the two operands in the destination operand. The status flags in the EFLAGS register indicate the result of the addition. This instruction can be combined with the LOCK prefix (see "LOCK—Assert LOCK# Signal Prefix" in Chapter 3, "Instruction Set Reference, A-L," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A) in a multiprocessing system to allow multiple processors to execute one DO loop.

The CMPXCHG (compare and exchange) and CMPXCHG8B (compare and exchange 8 bytes) instructions are used to synchronize operations in systems that use multiple processors. The CMPXCHG instruction requires three operands: a source operand in a register, another source operand in the EAX register, and a destination operand. If the values contained in the destination operand and the EAX register are equal, the destination operand is replaced with the value of the other source operand (the value not in the EAX register). Otherwise, the original

value of the destination operand is loaded in the EAX register. The status flags in the EFLAGS register reflect the result that would have been obtained by subtracting the destination operand from the value in the EAX register.

The CMPXCHG instruction is commonly used for testing and modifying semaphores. It checks to see if a semaphore is free. If the semaphore is free, it is marked allocated; otherwise it gets the ID of the current owner. This is all done in one uninterruptible operation. In a single-processor system, the CMPXCHG instruction eliminates the need to switch to protection level 0 (to disable interrupts) before executing multiple instructions to test and modify a semaphore.

For multiple processor systems, CMPXCHG can be combined with the LOCK prefix to perform the compare and exchange operation atomically. (See “Locked Atomic Operations” in Chapter 11, “Multiple-Processor Management,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, for more information on atomic operations.)

The CMPXCHG8B instruction also requires three operands: a 64-bit value in EDX:EAX, a 64-bit value in ECX:EBX, and a destination operand in memory. The instruction compares the 64-bit value in the EDX:EAX registers with the destination operand. If they are equal, the 64-bit value in the ECX:EBX registers is stored in the destination operand. If the EDX:EAX registers and the destination are not equal, the destination is loaded in the EDX:EAX registers. The CMPXCHG8B instruction can be combined with the LOCK prefix to perform the operation atomically.

7.3.1.3 Exchange Instructions in 64-Bit Mode

The CMPXCHG16B instruction is available in 64-bit mode only. It is an extension of the functionality provided by CMPXCHG8B that operates on 128-bits of data.

7.3.1.4 Stack Manipulation Instructions

The PUSH, POP, PUSHA (push all registers), and POPA (pop all registers) instructions move data to and from the stack. The PUSH instruction decrements the stack pointer (contained in the ESP register), then copies the source operand to the top of stack (see Figure 7-1). It operates on memory operands, immediate operands, and register operands (including segment registers). The PUSH instruction is commonly used to place parameters on the stack before calling a procedure. It can also be used to reserve space on the stack for temporary variables.

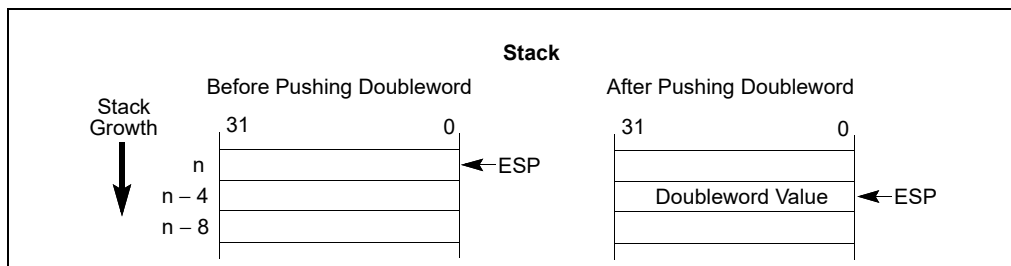


Figure 7-1. Operation of the PUSH Instruction

The PUSHA instruction saves the contents of the eight general-purpose registers on the stack (see Figure 7-2). This instruction simplifies procedure calls by reducing the number of instructions required to save the contents of the general-purpose registers. The registers are pushed on the stack in the following order: EAX, ECX, EDX, EBX, the initial value of ESP before EAX was pushed, EBP, ESI, and EDI.

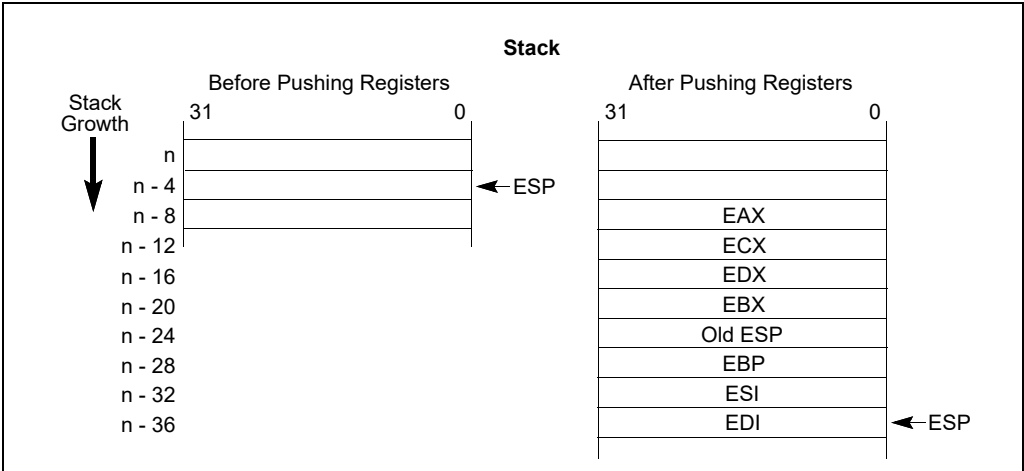


Figure 7-2. Operation of the PUSHA Instruction

The POP instruction copies the word or doubleword at the current top of stack (indicated by the ESP register) to the location specified with the destination operand. It then increments the ESP register to point to the new top of stack (see Figure 7-3). The destination operand may specify a general-purpose register, a segment register, or a memory location.

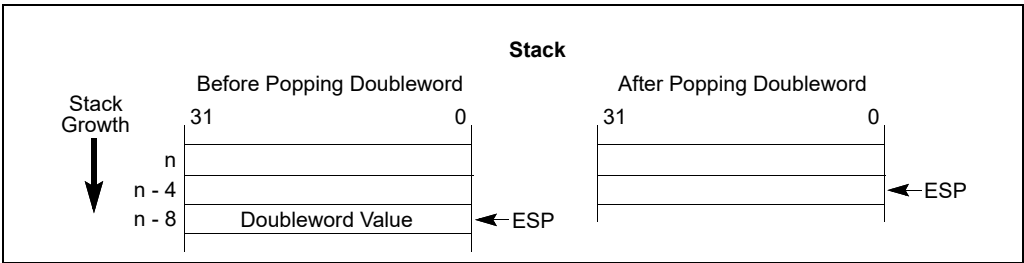


Figure 7-3. Operation of the POP Instruction

The POPA instruction reverses the effect of the PUSHA instruction. It pops the top eight words or doublewords from the top of the stack into the general-purpose registers, except for the ESP register (see Figure 7-4). If the operand-size attribute is 32, the doublewords on the stack are transferred to the registers in the following order: EDI, ESI, EBP, ignore doubleword, EBX, EDX, ECX, and EAX. The ESP register is restored by the action of popping the stack. If the operand-size attribute is 16, the words on the stack are transferred to the registers in the following order: DI, SI, BP, ignore word, BX, DX, CX, and AX.

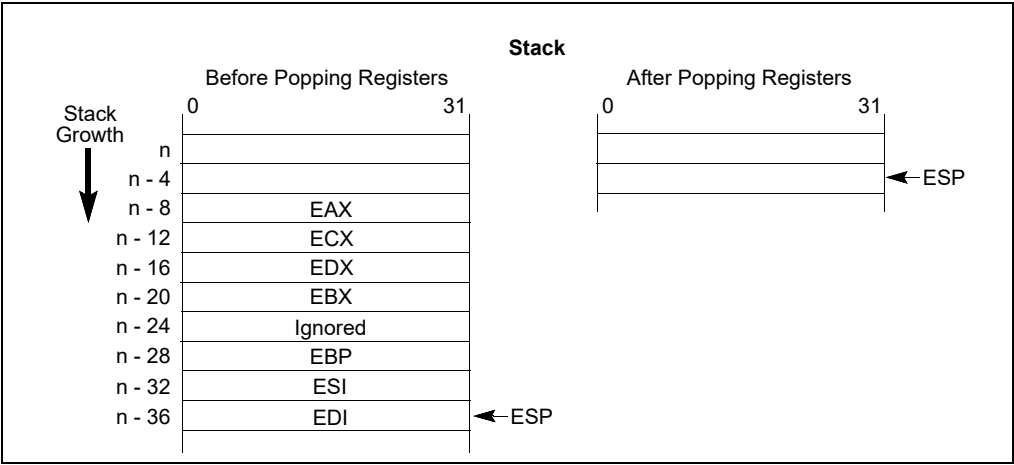


Figure 7-4. Operation of the POPA Instruction

7.3.1.5 Stack Manipulation Instructions in 64-Bit Mode

In 64-bit mode, the stack pointer size is 64 bits and cannot be overridden by an instruction prefix. In implicit stack references, address-size overrides are ignored. Pushes and pops of 32-bit values on the stack are not possible in 64-bit mode. 16-bit pushes and pops are supported by using the 66H operand-size prefix. PUSHAD, PUSHAD, POPA, and POPAD are not supported.

7.3.1.6 Type Conversion Instructions

The type conversion instructions convert bytes into words, words into doublewords, and doublewords into quadwords. These instructions are especially useful for converting integers to larger integer formats, because they perform sign extension (see Figure 7-5).

Two kinds of type conversion instructions are provided: simple conversion and move and convert.

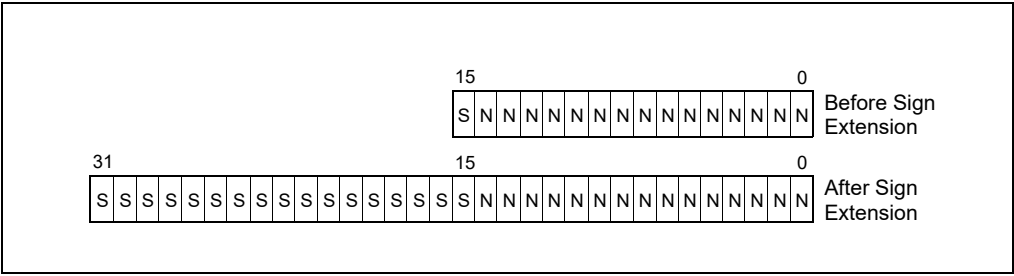


Figure 7-5. Sign Extension

Simple conversion — The CBW (convert byte to word), CWDE (convert word to doubleword extended), CWD (convert word to doubleword), and CDQ (convert doubleword to quadword) instructions perform sign extension to double the size of the source operand.

The CBW instruction copies the sign (bit 7) of the byte in the AL register into every bit position of the upper byte of the AX register. The CWDE instruction copies the sign (bit 15) of the word in the AX register into every bit position of the high word of the EAX register.

The CWD instruction copies the sign (bit 15) of the word in the AX register into every bit position in the DX register. The CDQ instruction copies the sign (bit 31) of the doubleword in the EAX register into every bit position in the EDX register. The CWD instruction can be used to produce a doubleword dividend from a word before a word division, and the CDQ instruction can be used to produce a quadword dividend from a doubleword before doubleword division.

Move with sign or zero extension — The MOVSX (move with sign extension) and MOVZX (move with zero extension) instructions move the source operand into a register then perform the sign extension.

The MOVSX instruction extends an 8-bit value to a 16-bit value or an 8-bit or 16-bit value to a 32-bit value by sign extending the source operand, as shown in Figure 7-5. The MOVZX instruction extends an 8-bit value to a 16-bit value or an 8-bit or 16-bit value to a 32-bit value by zero extending the source operand.

7.3.1.7 Type Conversion Instructions in 64-Bit Mode

The MOVSXD instruction operates on 64-bit data. It sign-extends a 32-bit value to 64 bits. This instruction is not encodable in non-64-bit modes.

7.3.2 Binary Arithmetic Instructions

Binary arithmetic instructions operate on 8-, 16-, and 32-bit numeric data encoded as signed or unsigned binary integers. The binary arithmetic instructions may also be used in algorithms that operate on decimal (BCD) values.

For the purpose of this discussion, these instructions are divided into subordinate subgroups of instructions that:

- Add and subtract.
- Increment and decrement.
- Compare and change signs.
- Multiply and divide.

7.3.2.1 Addition and Subtraction Instructions

The ADD (add integers), ADC (add integers with carry), SUB (subtract integers), and SBB (subtract integers with borrow) instructions perform addition and subtraction operations on signed or unsigned integer operands.

The ADD instruction computes the sum of two integer operands.

The ADC instruction computes the sum of two integer operands, plus 1 if the CF flag is set. This instruction is used to propagate a carry when adding numbers in stages.

The SUB instruction computes the difference of two integer operands.

The SBB instruction computes the difference of two integer operands, minus 1 if the CF flag is set. This instruction is used to propagate a borrow when subtracting numbers in stages.

7.3.2.2 Increment and Decrement Instructions

The INC (increment) and DEC (decrement) instructions add 1 to or subtract 1 from an unsigned integer operand, respectively. A primary use of these instructions is for implementing counters.

7.3.2.3 Increment and Decrement Instructions in 64-Bit Mode

The INC and DEC instructions are supported in 64-bit mode. However, some forms of INC and DEC (the register operand being encoded using register extension field in the MOD R/M byte) are not encodable in 64-bit mode because the opcodes are treated as REX prefixes.

7.3.2.4 Comparison and Sign Change Instructions

The CMP (compare) instruction computes the difference between two integer operands and updates the OF, SF, ZF, AF, PF, and CF flags according to the result. The source operands are not modified, nor is the result saved. The CMP instruction is commonly used in conjunction with a Jcc (jump) or SETcc (byte set on condition) instruction, with the latter instructions performing an action based on the result of a CMP instruction.

The NEG (negate) instruction subtracts a signed integer operand from zero. The effect of the NEG instruction is to change the sign of a two's complement operand while keeping its magnitude.

7.3.2.5 Multiplication and Division Instructions

The processor provides two multiply instructions, MUL (unsigned multiply) and IMUL (signed multiply), and two divide instructions, DIV (unsigned divide) and IDIV (signed divide).

The MUL instruction multiplies two unsigned integer operands. The result is computed to twice the size of the source operands (for example, if word operands are being multiplied, the result is a doubleword).

The IMUL instruction multiplies two signed integer operands. The result is computed to twice the size of the source operands; however, in some cases the result is truncated to the size of the source operands (see “IMUL—Signed Multiply” in Chapter 3, “Instruction Set Reference, A-L,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A).

The DIV instruction divides one unsigned operand by another unsigned operand and returns a quotient and a remainder.

The IDIV instruction is identical to the DIV instruction, except that IDIV performs a signed division.

7.3.3 Decimal Arithmetic Instructions

Decimal arithmetic can be performed by combining the binary arithmetic instructions ADD, SUB, MUL, and DIV (discussed in Section 7.3.2, “Binary Arithmetic Instructions”) with the decimal arithmetic instructions. The decimal arithmetic instructions are provided to carry out the following operations:

- To adjust the results of a previous binary arithmetic operation to produce a valid BCD result.
- To adjust the operands of a subsequent binary arithmetic operation so that the operation will produce a valid BCD result.

These instructions operate on both packed and unpacked BCD values. For the purpose of this discussion, the decimal arithmetic instructions are divided into subordinate subgroups of instructions that provide:

- Packed BCD adjustments.
- Unpacked BCD adjustments.

7.3.3.1 Packed BCD Adjustment Instructions

The DAA (decimal adjust after addition) and DAS (decimal adjust after subtraction) instructions adjust the results of operations performed on packed BCD integers (see Section 4.7, “BCD and Packed BCD Integers”). Adding two packed BCD values requires two instructions: an ADD instruction followed by a DAA instruction. The ADD instruction adds (binary addition) the two values and stores the result in the AL register. The DAA instruction then adjusts the value in the AL register to obtain a valid, 2-digit, packed BCD value and sets the CF flag if a decimal carry occurred as the result of the addition.

Likewise, subtracting one packed BCD value from another requires a SUB instruction followed by a DAS instruction. The SUB instruction subtracts (binary subtraction) one BCD value from another and stores the result in the AL register. The DAS instruction then adjusts the value in the AL register to obtain a valid, 2-digit, packed BCD value and sets the CF flag if a decimal borrow occurred as the result of the subtraction.

7.3.3.2 Unpacked BCD Adjustment Instructions

The AAA (ASCII adjust after addition), AAS (ASCII adjust after subtraction), AAM (ASCII adjust after multiplication), and AAD (ASCII adjust before division) instructions adjust the results of arithmetic operations performed on unpacked BCD values (see Section 4.7, “BCD and Packed BCD Integers”). All these instructions assume that the value to be adjusted is stored in the AL register or, in one instance, the AL and AH registers.

The AAA instruction adjusts the contents of the AL register following the addition of two unpacked BCD values. It converts the binary value in the AL register into a decimal value and stores the result in the AL register in unpacked BCD format (the decimal number is stored in the lower 4 bits of the register and the upper 4 bits are cleared). If a decimal carry occurred as a result of the addition, the CF flag is set and the contents of the AH register are incremented by 1.

The AAS instruction adjusts the contents of the AL register following the subtraction of two unpacked BCD values. Here again, a binary value is converted into an unpacked BCD value. If a borrow was required to complete the decimal subtract, the CF flag is set and the contents of the AH register are decremented by 1.

The AAM instruction adjusts the contents of the AL register following a multiplication of two unpacked BCD values. It converts the binary value in the AL register into a decimal value and stores the least significant digit of the result in the AL register (in unpacked BCD format) and the most significant digit, if there is one, in the AH register (also in unpacked BCD format).

The AAD instruction adjusts a two-digit BCD value so that when the value is divided with the DIV instruction, a valid unpacked BCD result is obtained. The instruction converts the BCD value in registers AH (most significant digit) and AL (least significant digit) into a binary value and stores the result in register AL. When the value in AL is divided by an unpacked BCD value, the quotient and remainder will be automatically encoded in unpacked BCD format.

7.3.4 Decimal Arithmetic Instructions in 64-Bit Mode

Decimal arithmetic instructions are not supported in 64-bit mode, they are either invalid or not encodable.

7.3.5 Logical Instructions

The logical instructions AND, OR, XOR (exclusive or), and NOT perform the standard Boolean operations for which they are named. The AND, OR, and XOR instructions require two operands; the NOT instruction operates on a single operand.

7.3.6 Shift and Rotate Instructions

The shift and rotate instructions rearrange the bits within an operand. For the purpose of this discussion, these instructions are further divided into subordinate subgroups of instructions that:

- Shift bits
- Double-shift bits (move them between operands)
- Rotate bits

7.3.6.1 Shift Instructions

The SAL (shift arithmetic left), SHL (shift logical left), SAR (shift arithmetic right), SHR (shift logical right) instructions perform an arithmetic or logical shift of the bits in a byte, word, or doubleword.

The SAL and SHL instructions perform the same operation (see Figure 7-6). They shift the source operand left by from 1 to 31 bit positions. Empty bit positions are cleared. The CF flag is loaded with the last bit shifted out of the operand.

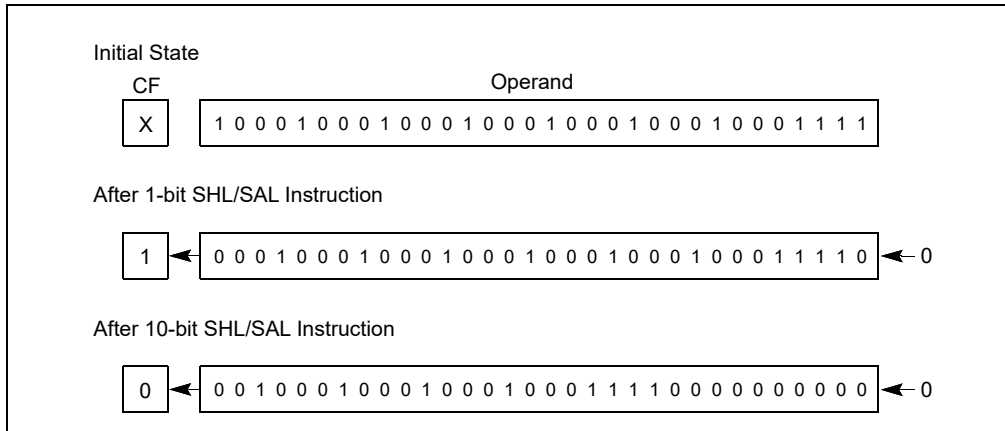


Figure 7-6. SHL/SAL Instruction Operation

The SHR instruction shifts the source operand right by from 1 to 31 bit positions (see Figure 7-7). As with the SHL/SAL instruction, the empty bit positions are cleared and the CF flag is loaded with the last bit shifted out of the operand.

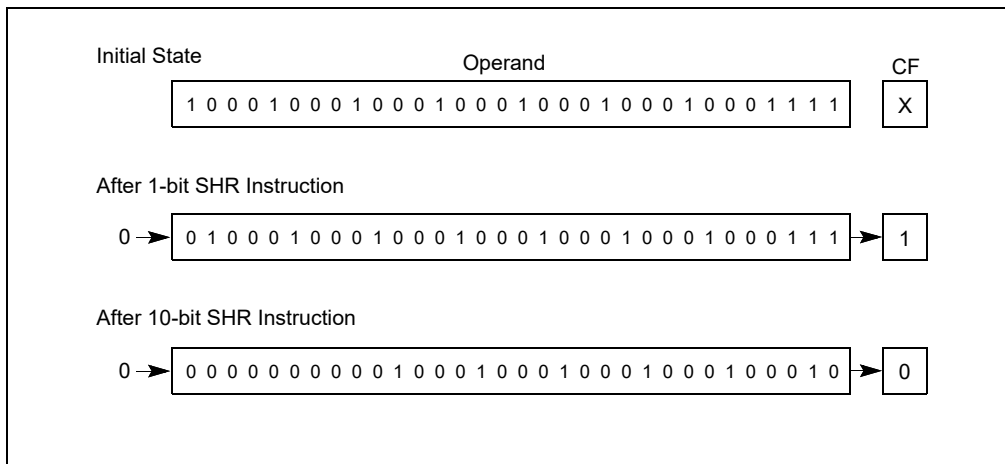


Figure 7-7. SHR Instruction Operation

The SAR instruction shifts the source operand right by from 1 to 31 bit positions (see Figure 7-8). This instruction differs from the SHR instruction in that it preserves the sign of the source operand by clearing empty bit positions if the operand is positive or setting the empty bits if the operand is negative. Again, the CF flag is loaded with the last bit shifted out of the operand.

The SAR and SHR instructions can also be used to perform division by powers of 2 (see “SAR/SAR/SHL/SHR—Shift Instructions” in Chapter 4, “Instruction Set Reference, M-U,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B).

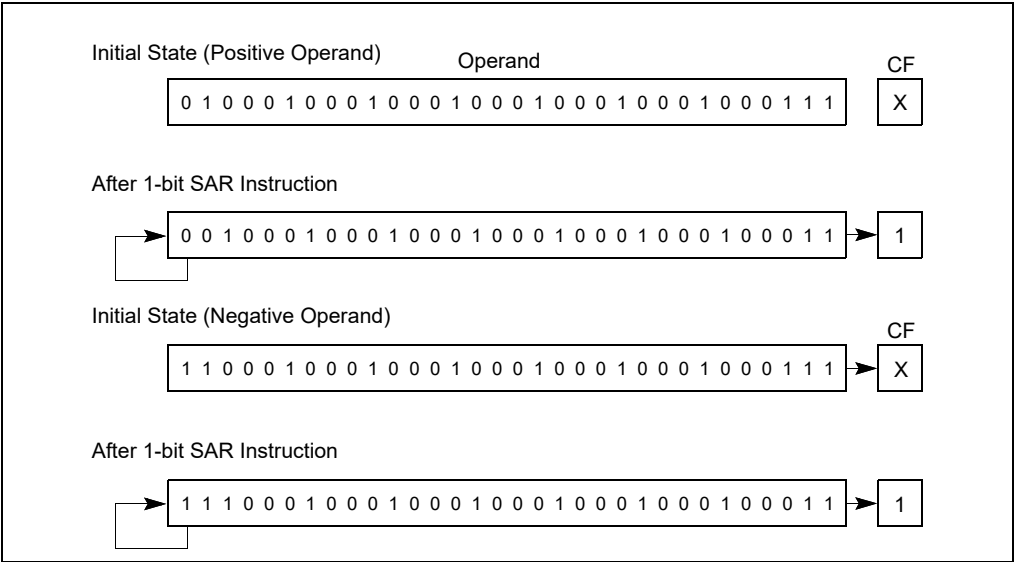


Figure 7-8. SAR Instruction Operation

7.3.6.2 Double-Shift Instructions

The SHLD (shift left double) and SHRD (shift right double) instructions shift a specified number of bits from one operand to another (see Figure 7-9). They are provided to facilitate operations on unaligned bit strings. They can also be used to implement a variety of bit string move operations.

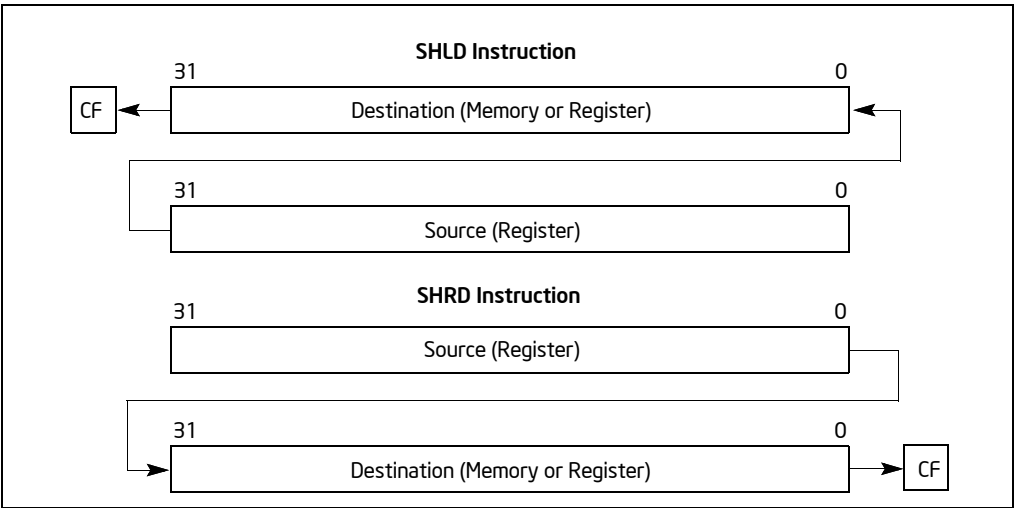


Figure 7-9. SHLD and SHRD Instruction Operations

The SHLD instruction shifts the bits in the destination operand to the left and fills the empty bit positions (in the destination operand) with bits shifted out of the source operand. The destination and source operands must be the same length (either words or doublewords). The shift count can range from 0 to 31 bits. The result of this shift operation is stored in the destination operand, and the source operand is not modified. The CF flag is loaded with the last bit shifted out of the destination operand.

The SHRD instruction operates the same as the SHLD instruction except bits are shifted to the right in the destination operand, with the empty bit positions filled with bits shifted out of the source operand.

7.3.6.3 Rotate Instructions

The ROL (rotate left), ROR (rotate right), RCL (rotate through carry left) and RCR (rotate through carry right) instructions rotate the bits in the destination operand out of one end and back through the other end (see Figure 7-10). Unlike a shift, no bits are lost during a rotation. The rotate count can range from 0 to 31.

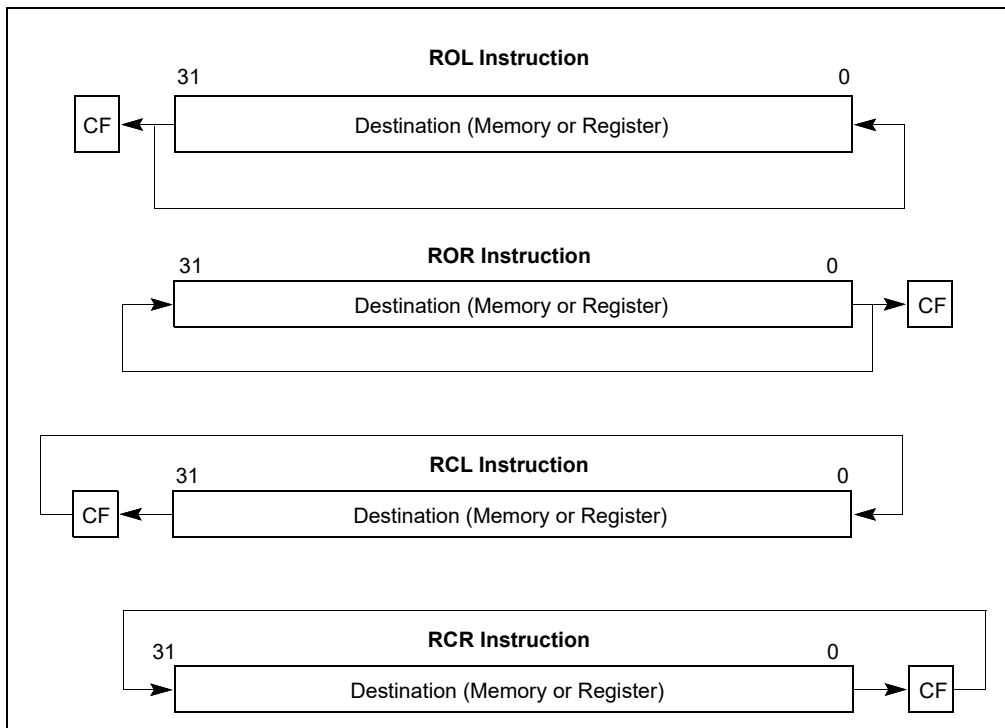


Figure 7-10. ROL, ROR, RCL, and RCR Instruction Operations

The ROL instruction rotates the bits in the operand to the left (toward more significant bit locations). The ROR instruction rotates the operand right (toward less significant bit locations).

The RCL instruction rotates the bits in the operand to the left, through the CF flag. This instruction treats the CF flag as a one-bit extension on the upper end of the operand. Each bit that exits from the most significant bit location of the operand moves into the CF flag. At the same time, the bit in the CF flag enters the least significant bit location of the operand.

The RCR instruction rotates the bits in the operand to the right through the CF flag.

For all the rotate instructions, the CF flag always contains the value of the last bit rotated out of the operand, even if the instruction does not use the CF flag as an extension of the operand. The value of this flag can then be tested by a conditional jump instruction (JC or JNC).

7.3.7 Bit and Byte Instructions

These instructions operate on bit or byte strings. For the purpose of this discussion, they are further divided into subordinate subgroups that:

- Test and modify a single bit.
- Scan a bit string.
- Set a byte given conditions.
- Test operands and report results.

7.3.7.1 Bit Test and Modify Instructions

The bit test and modify instructions (see Table 7-3) operate on a single bit, which can be in an operand. The location of the bit is specified as an offset from the least significant bit of the operand. When the processor identifies the bit to be tested and modified, it first loads the CF flag with the current value of the bit. Then it assigns a new value to the selected bit, as determined by the modify operation for the instruction.

Table 7-3. Bit Test and Modify Instructions

Instruction	Effect on CF Flag	Effect on Selected Bit
BT (Bit Test)	CF flag $\leftarrow$ Selected Bit	No effect
BTS (Bit Test and Set)	CF flag $\leftarrow$ Selected Bit	Selected Bit $\leftarrow$ 1
BTR (Bit Test and Reset)	CF flag $\leftarrow$ Selected Bit	Selected Bit $\leftarrow$ 0
BTC (Bit Test and Complement)	CF flag $\leftarrow$ Selected Bit	Selected Bit $\leftarrow$ NOT (Selected Bit)

7.3.7.2 Bit Scan Instructions

The BSF (bit scan forward) and BSR (bit scan reverse) instructions scan a bit string in a source operand for a set bit and store the bit index of the first set bit found in a destination register. The bit index is the offset from the least significant bit (bit 0) in the bit string to the first set bit. The BSF instruction scans the source operand low-to-high (from bit 0 of the source operand toward the most significant bit); the BSR instruction scans high-to-low (from the most significant bit toward the least significant bit).

7.3.7.3 Byte Set on Condition Instructions

The SETcc (set byte on condition) instructions set a destination-operand byte to 0 or 1, depending on the state of selected status flags (CF, OF, SF, ZF, and PF) in the EFLAGS register. The suffix (cc) added to the SET mnemonic determines the condition being tested for.

For example, the SETO instruction tests for overflow. If the OF flag is set, the destination byte is set to 1; if OF is clear, the destination byte is cleared to 0. Appendix B, “EFLAGS Condition Codes,” lists the conditions it is possible to test for with this instruction.

7.3.7.4 Test Instruction

The TEST instruction performs a logical AND of two operands and sets the SF, ZF, and PF flags according to the results. The flags can then be tested by the conditional jump or loop instructions or the SETcc instructions. The TEST instruction differs from the AND instruction in that it does not alter either of the operands.

7.3.8 Control Transfer Instructions

The processor provides both conditional and unconditional control transfer instructions to direct the flow of program execution. Conditional transfers are taken only for specified states of the status flags in the EFLAGS register. Unconditional control transfers are always executed.

For the purpose of this discussion, these instructions are further divided into subordinate subgroups that process:

- Unconditional transfers.
- Conditional transfers.
- Software interrupts.

7.3.8.1 Unconditional Transfer Instructions

The JMP, CALL, RET, INT, and IRET instructions transfer program control to another location (destination address) in the instruction stream. The destination can be within the same code segment (near transfer) or in a different code segment (far transfer).

Jump instruction — The JMP (jump) instruction unconditionally transfers program control to a destination instruction. The transfer is one-way; that is, a return address is not saved. A destination operand specifies the address (the instruction pointer) of the destination instruction. The address can be a **relative address** or an **absolute address**.

A **relative address** is a displacement (offset) with respect to the address in the EIP register. The destination address (a near pointer) is formed by adding the displacement to the address in the EIP register. The displacement is specified with a signed integer, allowing jumps either forward or backward in the instruction stream.

An **absolute address** is a offset from address 0 of a segment. It can be specified in either of the following ways:

- **An address in a general-purpose register** — This address is treated as a near pointer, which is copied into the EIP register. Program execution then continues at the new address within the current code segment.
- **An address specified using the standard addressing modes of the processor** — Here, the address can be a near pointer or a far pointer. If the address is for a near pointer, the address is translated into an offset and copied into the EIP register. If the address is for a far pointer, the address is translated into a segment selector (which is copied into the CS register) and an offset (which is copied into the EIP register).

In protected mode, the JMP instruction also allows jumps to a call gate, a task gate, and a task-state segment.

Call and return instructions — The CALL (call procedure) and RET (return from procedure) instructions allow a jump from one procedure (or subroutine) to another and a subsequent jump back (return) to the calling procedure.

The CALL instruction transfers program control from the current (or calling) procedure to another procedure (the called procedure). To allow a subsequent return to the calling procedure, the CALL instruction saves the current contents of the EIP register on the stack before jumping to the called procedure. The EIP register (prior to transferring program control) contains the address of the instruction following the CALL instruction. When this address is pushed on the stack, it is referred to as the **return instruction pointer** or **return address**.

The address of the called procedure (the address of the first instruction in the procedure being jumped to) is specified in a CALL instruction the same way as it is in a JMP instruction (see “Jump instruction” on page 7-15). The address can be specified as a relative address or an absolute address. If an absolute address is specified, it can be either a near or a far pointer.

The RET instruction transfers program control from the procedure currently being executed (the called procedure) back to the procedure that called it (the calling procedure). Transfer of control is accomplished by copying the return instruction pointer from the stack into the EIP register. Program execution then continues with the instruction pointed to by the EIP register.

The RET instruction has an optional operand, the value of which is added to the contents of the ESP register as part of the return operation. This operand allows the stack pointer to be incremented to remove parameters from the stack that were pushed on the stack by the calling procedure.

See Section 6.4, “Calling Procedures Using CALL and RET,” for more information on the mechanics of making procedure calls with the CALL and RET instructions.

Return from interrupt instruction — When the processor services an interrupt, it performs an implicit call to an interrupt-handling procedure. The IRET (return from interrupt) instruction returns program control from an interrupt handler to the interrupted procedure (that is, the procedure that was executing when the interrupt occurred). The IRET instruction performs a similar operation to the RET instruction (see “Call and return instructions” on page 7-15) except that it also restores the EFLAGS register from the stack. The contents of the EFLAGS register are automatically stored on the stack along with the return instruction pointer when the processor services an interrupt.

7.3.8.2 Conditional Transfer Instructions

The conditional transfer instructions execute jumps or loops that transfer program control to another instruction in the instruction stream if specified conditions are met. The conditions for control transfer are specified with a set of condition codes that define various states of the status flags (CF, ZF, OF, PF, and SF) in the EFLAGS register.

Conditional jump instructions — The Jcc (conditional) jump instructions transfer program control to a destination instruction if the conditions specified with the condition code (cc) associated with the instruction are satisfied (see Table 7-4). If the condition is not satisfied, execution continues with the instruction following the Jcc instruction. As with the JMP instruction, the transfer is one-way; that is, a return address is not saved.

Table 7-4. Conditional Jump Instructions

Instruction Mnemonic	Condition (Flag States)	Description
Unsigned Conditional Jumps		
JA/JNBE	(CF or ZF) = 0	Above/not below or equal
JAЕ/JNB	CF = 0	Above or equal/not below
JB/JNAЕ	CF = 1	Below/not above or equal
JBE/JNA	(CF or ZF) = 1	Below or equal/not above
JC	CF = 1	Carry
JE/JZ	ZF = 1	Equal/zero
JNC	CF = 0	Not carry
JNE/JNZ	ZF = 0	Not equal/not zero
JNP/JPO	PF = 0	Not parity/parity odd
JP/JPE	PF = 1	Parity/parity even
JCXZ	CX = 0	Register CX is zero
JECXZ	ECX = 0	Register ECX is zero
Signed Conditional Jumps		
JG/JNLE	((SF xor OF) or ZF) = 0	Greater/not less or equal
JGE/JNL	(SF xor OF) = 0	Greater or equal/not less
JL/JNGE	(SF xor OF) = 1	Less/not greater or equal
JLE/JNG	((SF xor OF) or ZF) = 1	Less or equal/not greater
JNO	OF = 0	Not overflow
JNS	SF = 0	Not sign (non-negative)
JO	OF = 1	Overflow
JS	SF = 1	Sign (negative)

The destination operand specifies a relative address (a signed offset with respect to the address in the EIP register) that points to an instruction in the current code segment. The Jcc instructions do not support far transfers; however, far transfers can be accomplished with a combination of a Jcc and a JMP instruction (see “Jcc—Jump if Condition Is Met” in Chapter 3, “Instruction Set Reference, A-L,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A).

Table 7-4 shows the mnemonics for the Jcc instructions and the conditions being tested for each instruction. The condition code mnemonics are appended to the letter “J” to form the mnemonic for a Jcc instruction. The instructions are divided into two groups: unsigned and signed conditional jumps. These groups correspond to the results of operations performed on unsigned and signed integers respectively. Those instructions listed as pairs (for example, JA/JNBE) are alternate names for the same instruction. Assemblers provide alternate names to make it easier to read program listings.

The JCXZ and JECXZ instructions test the CX and ECX registers, respectively, instead of one or more status flags. See “Jump if zero instructions” on page 7-17 for more information about these instructions.

Loop instructions — The LOOP, LOOPE (loop while equal), LOOPZ (loop while zero), LOOPNE (loop while not equal), and LOOPNZ (loop while not zero) instructions are conditional jump instructions that use the value of the ECX register as a count for the number of times to execute a loop. All the loop instructions decrement the count in the ECX register each time they are executed and terminate a loop when zero is reached. The LOOPE, LOOPZ, LOOPNE, and LOOPNZ instructions also accept the ZF flag as a condition for terminating the loop before the count reaches zero.

The LOOP instruction decrements the contents of the ECX register (or the CX register, if the address-size attribute is 16), then tests the register for the loop-termination condition. If the count in the ECX register is non-zero, program control is transferred to the instruction address specified by the destination operand. The destination

operand is a relative address (that is, an offset relative to the contents of the EIP register), and it generally points to the first instruction in the block of code that is to be executed in the loop. When the count in the ECX register reaches zero, program control is transferred to the instruction immediately following the LOOP instruction, which terminates the loop. If the count in the ECX register is zero when the LOOP instruction is first executed, the register is pre-decremented to FFFFFFFFH, causing the loop to be executed 2^{32} times.

The LOOPE and LOOPZ instructions perform the same operation (they are mnemonics for the same instruction). These instructions operate the same as the LOOP instruction, except that they also test the ZF flag.

If the count in the ECX register is not zero and the ZF flag is set, program control is transferred to the destination operand. When the count reaches zero or the ZF flag is clear, the loop is terminated by transferring program control to the instruction immediately following the LOOPE/LOOPZ instruction.

The LOOPNE and LOOPNZ instructions (mnemonics for the same instruction) operate the same as the LOOPE/LOOPZ instructions, except that they terminate the loop if the ZF flag is set.

Jump if zero instructions — The JECXZ (jump if ECX zero) instruction jumps to the location specified in the destination operand if the ECX register contains the value zero. This instruction can be used in combination with a loop instruction (LOOP, LOOPE, LOOPZ, LOOPNE, or LOOPNZ) to test the ECX register prior to beginning a loop. As described in “Loop instructions” on page 7-16, the loop instructions decrement the contents of the ECX register before testing for zero. If the value in the ECX register is zero initially, it will be decremented to FFFFFFFFH on the first loop instruction, causing the loop to be executed 2^{32} times. To prevent this problem, a JECXZ instruction can be inserted at the beginning of the code block for the loop, causing a jump out of the loop if the ECX register count is initially zero. When used with repeated string scan and compare instructions, the JECXZ instruction can determine whether the loop terminated because the count reached zero or because the scan or compare conditions were satisfied.

The JCXZ (jump if CX is zero) instruction operates the same as the JECXZ instruction when the 16-bit address-size attribute is used. Here, the CX register is tested for zero.

7.3.8.3 Control Transfer Instructions in 64-Bit Mode

In 64-bit mode, the operand size for all near branches (CALL, RET, JCC, JCXZ, JMP, and LOOP) is forced to 64 bits. The listed instructions update the 64-bit RIP without need for a REX operand-size prefix.

Near branches in the following operations are forced to 64-bits (regardless of operand size prefixes):

- Truncation of the size of the instruction pointer.
- Size of a stack pop or push, due to CALL or RET.
- Size of a stack-pointer increment or decrement, due to CALL or RET.
- Indirect-branch operand size.

Note that the displacement field for relative branches is still limited to 32 bits and the address size for near branches is not forced.

Address size determines the register size (CX/ECX/RCX) used for JCXZ and LOOP. It also impacts the address calculation for memory indirect branches. Addresses size is 64 bits by default, although it can be over-ridden to 32 bits (using a prefix).

7.3.8.4 Software Interrupt Instructions

The INT *n* (software interrupt), INTO (interrupt on overflow), and BOUND (detect value out of range) instructions allow a program to explicitly raise a specified interrupt or exception, which in turn causes the handler routine for the interrupt or exception to be called.

The INT *n* instruction can raise any of the processor’s interrupts or exceptions by encoding the vector of the interrupt or exception in the instruction. This instruction can be used to support software generated interrupts or to test the operation of interrupt and exception handlers.

The IRET (return from interrupt) instruction returns program control from an interrupt handler to the interrupted procedure. The IRET instruction performs a similar operation to the RET instruction.

The CALL (call procedure) and RET (return from procedure) instructions allow a jump from one procedure to another and a subsequent return to the calling procedure. EFLAGS register contents are automatically stored on the stack along with the return instruction pointer when the processor services an interrupt.

The INTO instruction raises the overflow exception if the OF flag is set. If the flag is clear, execution continues without raising the exception. This instruction allows software to access the overflow exception handler explicitly to check for overflow conditions.

The BOUND instruction compares a signed value against upper and lower bounds, and raises the “BOUND range exceeded” exception if the value is less than the lower bound or greater than the upper bound. This instruction is useful for operations such as checking an array index to make sure it falls within the range defined for the array.

7.3.8.5 Software Interrupt Instructions in 64-Bit Mode and Compatibility Mode

In 64-bit mode, the stack size is 8 bytes wide. IRET must pop 8-byte items off the stack. SS:RSP pops unconditionally. BOUND is not supported.

In compatibility mode, SS:RSP is popped only if the CPL changes.

7.3.9 String Operations

The GP instructions includes a set of **string instructions** that are designed to access large data structures; these are introduced in Section 7.3.9.1. Section 7.3.9.2 describes how REP prefixes can be used with these instructions to perform more complex **repeated string operations**. Certain processors optimize repeated string operations with **fast-string operation**, as described in Section 7.3.9.3. Section 7.3.9.4 explains how string operations can be used in 64-bit mode.

7.3.9.1 String Instructions

The MOVS (Move String), CMPS (Compare string), SCAS (Scan string), LODS (Load string), and STOS (Store string) instructions permit large data structures, such as alphanumeric character strings, to be moved and examined in memory. These instructions operate on individual elements in a string, which can be a byte, word, or doubleword. The string elements to be operated on are identified with the ESI (source string element) and EDI (destination string element) registers. Both of these registers contain absolute addresses (offsets into a segment) that point to a string element.

By default, the ESI register addresses the segment identified with the DS segment register. A segment-override prefix allows the ESI register to be associated with the CS, SS, ES, FS, or GS segment register. The EDI register addresses the segment identified with the ES segment register; no segment override is allowed for the EDI register. The use of two different segment registers in the string instructions permits operations to be performed on strings located in different segments. Or by associating the ESI register with the ES segment register, both the source and destination strings can be located in the same segment. (This latter condition can also be achieved by loading the DS and ES segment registers with the same segment selector and allowing the ESI register to default to the DS register.)

The MOVS instruction moves the string element addressed by the ESI register to the location addressed by the EDI register. The assembler recognizes three “short forms” of this instruction, which specify the size of the string to be moved: MOVSB (move byte string), MOVSW (move word string), and MOVSD (move doubleword string).

The CMPS instruction subtracts the destination string element from the source string element and updates the status flags (CF, ZF, OF, SF, PF, and AF) in the EFLAGS register according to the results. Neither string element is written back to memory. The assembler recognizes three “short forms” of the CMPS instruction: CMPSB (compare byte strings), CMPSW (compare word strings), and CMPSD (compare doubleword strings).

The SCAS instruction subtracts the destination string element from the contents of the EAX, AX, or AL register (depending on operand length) and updates the status flags according to the results. The string element and register contents are not modified. The following “short forms” of the SCAS instruction specify the operand length: SCASB (scan byte string), SCASW (scan word string), and SCASD (scan doubleword string).

The LODS instruction loads the source string element identified by the ESI register into the EAX register (for a doubleword string), the AX register (for a word string), or the AL register (for a byte string). The “short forms” for

this instruction are LODSB (load byte string), LODSW (load word string), and LODSD (load doubleword string). This instruction is usually used in a loop, where other instructions process each element of the string after they are loaded into the target register.

The STOS instruction stores the source string element from the EAX (doubleword string), AX (word string), or AL (byte string) register into the memory location identified with the EDI register. The “short forms” for this instruction are STOSB (store byte string), STOSW (store word string), and STOSD (store doubleword string). This instruction is also normally used in a loop. Here a string is commonly loaded into the register with a LODS instruction, operated on by other instructions, and then stored again in memory with a STOS instruction.

The I/O instructions (see Section 7.3.10, “I/O Instructions”) also perform operations on strings in memory.

7.3.9.2 Repeated String Operations

Each of the string instructions described in Section 7.3.9.1 perform one iteration of a string operation. To operate on strings longer than a doubleword, the string instructions can be combined with a repeat prefix (REP) to create a repeating instruction or be placed in a loop.

When used in string instructions, the ESI and EDI registers are automatically incremented or decremented after each iteration of an instruction to point to the next element (byte, word, or doubleword) in the string. String operations can thus begin at higher addresses and work toward lower ones, or they can begin at lower addresses and work toward higher ones. The DF flag in the EFLAGS register controls whether the registers are incremented (DF = 0) or decremented (DF = 1). The STD and CLD instructions set and clear this flag, respectively.

The following repeat prefixes can be used in conjunction with a count in the ECX register to cause a string instruction to repeat:

- **REP** — Repeat while the ECX register not zero.
- **REPE/REPZ** — Repeat while the ECX register not zero and the ZF flag is set.
- **REPNE/REPNZ** — Repeat while the ECX register not zero and the ZF flag is clear.

When a string instruction has a repeat prefix, the operation executes until one of the termination conditions specified by the prefix is satisfied. The REPE/REPZ and REPNE/REPNZ prefixes are used only with the CMPS and SCAS instructions. Also, note that a REP STOS instruction is the fastest way to initialize a large block of memory.

7.3.9.3 Fast-String Operation

To improve performance, more recent processors support modifications to the processor’s operation during the string store operations initiated with the MOVS, MOVSB, STOS, and STOSB instructions. This optimized operation, called **fast-string operation**, is used when the execution of one of those instructions meets certain initial conditions (see below). Instructions using fast-string operation effectively operate on the string in groups that may include multiple elements of the native data size (byte, word, doubleword, or quadword). With fast-string operation, the processor recognizes interrupts and data breakpoints only on boundaries between these groups. Fast-string operation is used only if the source and destination addresses both use either the WB or WC memory types.

The initial conditions for fast-string operation are implementation-specific and may vary with the native string size. Examples of parameters that may impact the use of fast-string operation include the following:

- The alignment indicated in the EDI and ESI alignment registers.
- The address order of the string operation.
- The value of the initial operation counter (ECX).
- The difference between the source and destination addresses.

NOTE

Initial conditions for fast-string operation in future Intel 64 or IA-32 processor families may differ from above. The Intel® 64 and IA-32 Architectures Optimization Reference Manual may contain model-specific information.

Software can disable fast-string operation by clearing the fast-string-enable bit (bit 0) of IA32_MISC_ENABLE MSR. However, Intel recommends that system software always enable fast-string operation.

When fast-string operation is enabled (because IA32_MISC_ENABLE[0] = 1), some processors may further enhance the operation of the REP MOVSB and REP STOSB instructions. A processor supports these enhancements if CPUID.07H:EBX[9] is 1. The Intel® 64 and IA-32 Architectures Optimization Reference Manual may include model-specific recommendations for use of these enhancements.

The stores produced by fast-string operation may appear to execute out of order **or be repeated**. Software dependent upon sequential store ordering should not use string operations for the entire data structure to be stored. Data and semaphores should be separated. Order-dependent code should write to a discrete semaphore variable after any string operations to allow correctly ordered data to be seen by all processors. Atomicity of load and store operations is guaranteed only for native data elements of the string with native data size, and only if they are included in a single cache line. See Section 11.2.4, “Fast-String Operation and Out-of-Order Stores,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

7.3.9.4 String Operations in 64-Bit Mode

The behavior of MOVSB (Move String), CMPS (Compare string), SCAS (Scan string), LODS (Load string), and STOS (Store string) instructions in 64-bit mode is similar to their behavior in non-64-bit modes, with the following differences:

- The source operand is specified by RSI or DS:ESI, depending on the address size attribute of the operation.
- The destination operand is specified by RDI or DS:EDI, depending on the address size attribute of the operation.
- Operation on 64-bit data is supported by using the REX.W prefix.

When using REP prefixes for string operations in 64-bit mode, the repeat count is specified by RCX or ECX (depending on the address size attribute of the operation). The default address size is 64 bits.

7.3.10 I/O Instructions

The IN (input from port to register), INS (input from port to string), OUT (output from register to port), and OUTS (output string to port) instructions move data between the processor’s I/O ports and either a register or memory.

The register I/O instructions (IN and OUT) move data between an I/O port and the EAX register (32-bit I/O), the AX register (16-bit I/O), or the AL (8-bit I/O) register. The I/O port being read or written to is specified with an immediate operand or an address in the DX register.

The block I/O instructions (INS and OUTS) instructions move blocks of data (strings) between an I/O port and memory. These instructions operate similar to the string instructions (see Section 7.3.9, “String Operations”). The ESI and EDI registers are used to specify string elements in memory and the repeat prefix (REP) is used to repeat the instructions to implement block moves. The assembler recognizes the following alternate mnemonics for these instructions: INSB (input byte), INSW (input word), and INSD (input doubleword), and OUTSB (output byte), OUTSW (output word), and OUTSD (output doubleword).

The INS and OUTS instructions use an address in the DX register to specify the I/O port to be read or written to.

7.3.11 I/O Instructions in 64-Bit Mode

For I/O instructions to and from memory, the differences in 64-bit mode are:

- The source operand is specified by RSI or DS:ESI, depending on the address size attribute of the operation.
- The destination operand is specified by RDI or DS:EDI, depending on the address size attribute of the operation.
- Operation on 64-bit data is not encodable and REX prefixes are silently ignored.

7.3.12 Enter and Leave Instructions

The ENTER and LEAVE instructions provide machine-language support for procedure calls in block-structured languages, such as C and Pascal. These instructions and the call and return mechanism that they support are described in detail in Section 6.6, “Procedure Calls for Block-Structured Languages.”

7.3.13 Flag Control (EFLAG) Instructions

The Flag Control (EFLAG) instructions allow the state of selected flags in the EFLAGS register to be read or modified. For the purpose of this discussion, these instructions are further divided into subordinate subgroups of instructions that manipulate:

- Carry and direction flags.
- The EFLAGS register.
- Interrupt flags.

7.3.13.1 Carry and Direction Flag Instructions

The STC (set carry flag), CLC (clear carry flag), and CMC (complement carry flag) instructions allow the CF flag in the EFLAGS register to be modified directly. They are typically used to initialize the CF flag to a known state before an instruction that uses the flag in an operation is executed. They are also used in conjunction with the rotate-with-carry instructions (RCL and RCR).

The STD (set direction flag) and CLD (clear direction flag) instructions allow the DF flag in the EFLAGS register to be modified directly. The DF flag determines the direction in which index registers ESI and EDI are stepped when executing string processing instructions. If the DF flag is clear, the index registers are incremented after each iteration of a string instruction; if the DF flag is set, the registers are decremented.

7.3.13.2 EFLAGS Transfer Instructions

The EFLAGS transfer instructions allow groups of flags in the EFLAGS register to be copied to a register or memory or be loaded from a register or memory.

The LAHF (load AH from flags) and SAHF (store AH into flags) instructions operate on five of the EFLAGS status flags (SF, ZF, AF, PF, and CF). The LAHF instruction copies the status flags to bits 7, 6, 4, 2, and 0 of the AH register, respectively. The contents of the remaining bits in the register (bits 5, 3, and 1) are unaffected, and the contents of the EFLAGS register remain unchanged. The SAHF instruction copies bits 7, 6, 4, 2, and 0 from the AH register into the SF, ZF, AF, PF, and CF flags, respectively in the EFLAGS register.

The PUSHF (push flags), PUSHFD (push flags double), POPF (pop flags), and POPFD (pop flags double) instructions copy the flags in the EFLAGS register to and from the stack. The PUSHF instruction pushes the lower word of the EFLAGS register onto the stack (see Figure 7-11). The PUSHFD instruction pushes the entire EFLAGS register onto the stack (with the RF and VM flags read as clear).

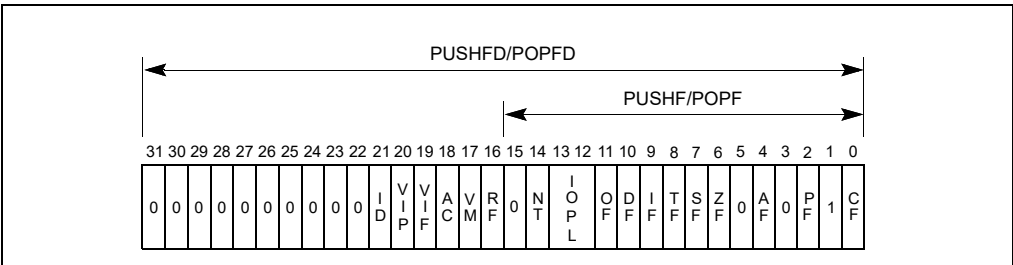


Figure 7-11. Flags Affected by the PUSHF, POPF, PUSHFD, and POPFD Instructions

The POPF instruction pops a word from the stack into the EFLAGS register. Only bits 11, 10, 8, 7, 6, 4, 2, and 0 of the EFLAGS register are affected with all uses of this instruction. If the current privilege level (CPL) of the current

code segment is 0 (most privileged), the IOPL bits (bits 13 and 12) also are affected. If the I/O privilege level (IOPL) is greater than or equal to the CPL, numerically, the IF flag (bit 9) also is affected.

The POPFD instruction pops a doubleword into the EFLAGS register. This instruction can change the state of the AC bit (bit 18) and the ID bit (bit 21), as well as the bits affected by a POPF instruction. The restrictions for changing the IOPL bits and the IF flag that were given for the POPF instruction also apply to the POPFD instruction.

7.3.13.3 Interrupt Flag Instructions

The STI (set interrupt flag) and CLI (clear interrupt flag) instructions allow the interrupt IF flag in the EFLAGS register to be modified directly. The IF flag controls the servicing of hardware-generated interrupts (those received at the processor's INTR pin). If the IF flag is set, the processor services hardware interrupts; if the IF flag is clear, hardware interrupts are masked.

The ability to execute these instructions depends on the operating mode of the processor and the current privilege level (CPL) of the program or task attempting to execute these instructions.

7.3.14 Flag Control (RFLAG) Instructions in 64-Bit Mode

In 64-bit mode, the LAHF and SAHF instructions are supported if CPUID.80000001H:ECX.LAHF_SAHF_64[0] = 1.

PUSHF and POPF behave the same in 64-bit mode as in non-64-bit mode. PUSHFD always pushes 64-bit RFLAGS onto the stack (with the RF and VM flags read as clear). POPFD always pops a 64-bit value from the top of the stack and loads the lower 32 bits into RFLAGS. It then zero extends the upper bits of RFLAGS.

7.3.15 Segment Register Instructions

The processor provides a variety of instructions that address the segment registers of the processor directly. These instructions are only used when an operating system or executive is using the segmented or the real-address mode memory model.

For the purpose of this discussion, these instructions are divided into subordinate subgroups of instructions that allow:

- Segment-register load and store.
- Far control transfers.
- Software interrupt calls.
- Handling of far pointers.

7.3.15.1 Segment-Register Load and Store Instructions

The MOV instruction (introduced in Section 7.3.1.1, "General Data Movement Instructions") and the PUSH and POP instructions (introduced in Section 7.3.1.4, "Stack Manipulation Instructions") can transfer 16-bit segment selectors to and from segment registers (DS, ES, FS, GS, and SS). The transfers are always made to or from a segment register and a general-purpose register or memory. Transfers between segment registers are not supported.

The POP and MOV instructions cannot place a value in the CS register. Only the far control-transfer versions of the JMP, CALL, and RET instructions (see Section 7.3.15.2, "Far Control Transfer Instructions") affect the CS register directly.

7.3.15.2 Far Control Transfer Instructions

The JMP and CALL instructions (see Section 7.3.8, "Control Transfer Instructions") both accept a far pointer as a destination to transfer program control to a segment other than the segment currently being pointed to by the CS register. When a far call is made with the CALL instruction, the current values of the EIP and CS registers are both pushed on the stack.

The RET instruction (see "Call and return instructions" on page 7-15) can be used to execute a far return. Here, program control is transferred from a code segment that contains a called procedure back to the code segment that

contained the calling procedure. The RET instruction restores the values of the CS and EIP registers for the calling procedure from the stack.

7.3.15.3 Software Interrupt Instructions

The software interrupt instructions INT, INTO, and IRET (see Section 7.3.8.4, “Software Interrupt Instructions”) can also call and return from interrupt and exception handler procedures that are located in a code segment other than the current code segment. With these instructions, however, the switching of code segments is handled transparently from the application program.

7.3.15.4 Load Far Pointer Instructions

The load far pointer instructions LDS (load far pointer using DS), LES (load far pointer using ES), LFS (load far pointer using FS), LGS (load far pointer using GS), and LSS (load far pointer using SS) load a far pointer from memory into a segment register and a general-purpose general register. The segment selector part of the far pointer is loaded into the selected segment register and the offset is loaded into the selected general-purpose register.

7.3.16 Miscellaneous Instructions

The following instructions perform operations that are of interest to applications programmers. For the purpose of this discussion, these instructions are further divided into subordinate subgroups of instructions that provide for:

- Address computations.
- Table lookup.
- Processor identification.
- NOP and undefined instruction entry.

7.3.16.1 Address Computation Instruction

The LEA (load effective address) instruction computes the effective address in memory (offset within a segment) of a source operand and places it in a general-purpose register. This instruction can interpret any of the processor’s addressing modes and can perform any indexing or scaling that may be needed. It is especially useful for initializing the ESI or EDI registers before the execution of string instructions or for initializing the EBX register before an XLAT instruction.

7.3.16.2 Table Lookup Instructions

The XLAT and XLATB (table lookup) instructions replace the contents of the AL register with a byte read from a translation table in memory. The initial value in the AL register is interpreted as an unsigned index into the translation table. This index is added to the contents of the EBX register (which contains the base address of the table) to calculate the address of the table entry. These instructions are used for applications such as converting character codes from one alphabet into another (for example, an ASCII code could be used to look up its EBCDIC equivalent in a table).

7.3.16.3 Processor Identification Instruction

The CPUID (processor identification) instruction returns information about the processor on which the instruction is executed.

7.3.16.4 No-Operation and Undefined Instructions

The NOP (no operation) instruction increments the EIP register to point at the next instruction, but affects nothing else.

The UD (undefined) instruction generates an invalid opcode exception. Intel reserves the opcode for this instruction for this function. The instruction is provided to allow software to test an invalid opcode exception handler.

7.3.17 Random Number Generator Instructions

The instructions for generating random numbers to comply with NIST SP800-90A, SP800-90B, and SP800-90C standards are described in this section.

7.3.17.1 RDRAND

The RDRAND instruction returns a random number. All Intel processors that support the RDRAND instruction indicate the availability of the RDRAND instruction via reporting CPUID.01H:ECX.RDRAND[30] = 1.

RDRAND returns random numbers that are supplied by a cryptographically secure, deterministic random bit generator DRBG. The DRBG is designed to meet the NIST SP 800-90A standard. The DRBG is re-seeded frequently from an on-chip non-deterministic entropy source to guarantee data returned by RDRAND is statistically uniform, non-periodic and non-deterministic.

In order for the hardware design to meet its security goals, the random number generator continuously tests itself and the random data it is generating. Runtime failures in the random number generator circuitry or statistically anomalous data occurring by chance will be detected by the self test hardware and flag the resulting data as being bad. In such extremely rare cases, the RDRAND instruction will return no data instead of bad data.

Under heavy load, with multiple cores executing RDRAND in parallel, it is possible, though unlikely, for the demand of random numbers by software processes/threads to exceed the rate at which the random number generator hardware can supply them. This will lead to the RDRAND instruction returning no data transitorily. The RDRAND instruction indicates the occurrence of this rare situation by clearing the CF flag.

The RDRAND instruction returns with the carry flag set (CF = 1) to indicate valid data is returned. It is recommended that software using the RDRAND instruction to get random numbers retry for a limited number of iterations while RDRAND returns CF = 0 and complete when valid data is returned, indicated with CF = 1. This will deal with transitory underflows. A retry limit should be employed to prevent a hard failure in the RNG (expected to be extremely rare) leading to a busy loop in software.

The intrinsic primitive for RDRAND is defined to address software's need for the common cases (CF = 1) and the rare situations (CF = 0). The intrinsic primitive returns a value that reflects the value of the carry flag returned by the underlying RDRAND instruction. The example below illustrates the recommended usage of an RDRAND intrinsic in a utility function, a loop to fetch a 64 bit random value with a retry count limit of 10. A C implementation might be written as follows:

```
-----
#define SUCCESS 1
#define RETRY_LIMIT_EXCEEDED 0
#define RETRY_LIMIT 10

int get_random_64(unsigned __int64 * arand)
{int i ;
  for (i = 0; i < RETRY_LIMIT; i++) {
    if(_rdrand64_step(arand) ) return SUCCESS;
  }
  return RETRY_LIMIT_EXCEEDED;
}
-----
```

7.3.17.2 RDSEED

The RDSEED instruction returns a random number. All Intel processors that support the RDSEED instruction indicate the availability of the RDSEED instruction via reporting CPUID.07H.00H:EBX.RDSEED[18] = 1.

RDSEED returns random numbers that are supplied by a cryptographically secure, enhanced non-deterministic random bit generator (Enhanced NRBG). The NRBG is designed to meet the NIST SP 800-90B and NIST SP800-90C standards.

In order for the hardware design to meet its security goals, the random number generator continuously tests itself and the random data it is generating. Runtime failures in the random number generator circuitry or statistically anomalous data occurring by chance will be detected by the self test hardware and flag the resulting data as being bad. In such extremely rare cases, the RDSEED instruction will return no data instead of bad data.

Under heavy load, with multiple cores executing RDSEED in parallel, it is possible for the demand of random numbers by software processes/threads to exceed the rate at which the random number generator hardware can supply them. This will lead to the RDSEED instruction returning no data transitorily. The RDSEED instruction indicates the occurrence of this situation by clearing the CF flag.

The RDSEED instruction returns with the carry flag set ($CF = 1$) to indicate valid data is returned. It is recommended that software using the RDSEED instruction to get random numbers retry for a limited number of iterations while RDSEED returns $CF = 0$ and complete when valid data is returned, indicated with $CF = 1$. This will deal with transitory underflows. A retry limit should be employed to prevent a hard failure in the NRBG (expected to be extremely rare) leading to a busy loop in software.

The intrinsic primitive for RDSEED is defined to address software's need for the common cases ($CF = 1$) and the rare situations ($CF = 0$). The intrinsic primitive returns a value that reflects the value of the carry flag returned by the underlying RDSEED instruction.

4. Updates to Chapter 18, Volume 1

Change bars and text show changes to Chapter 18 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

Changes to this chapter:

- In Section 18.2.3, "Supervisor Shadow Stack Token", clarified the behavior for shadow-stack support provided for call gates, IDT event delivery, and IRET; and noted that does not apply when Flexible Return and Event Delivery transitions are enabled.

18.1 INTRODUCTION

Return-oriented programming (ROP), and similarly CALL/JMP-oriented programming (COP/JOP), have been the prevalent attack methodologies for stealth exploit writers targeting vulnerabilities in programs. These attack methodologies have the common elements:

- A code module with execution privilege and contain small snippets of code sequence with the characteristic: at least one instruction in the sequence being a control transfer instruction that depends on data either in the return stack or in a register for the target address.
- Diverting the control flow instruction (e.g., RET, CALL, JMP) from its original target address to a new target (via modification in the data stack or in the register).

Control-Flow Enforcement Technology (CET) provides the following capabilities to defend against ROP/COP/JOP style control-flow subversion attacks:

- Shadow stack: Return address protection to defend against ROP.
- Indirect branch tracking: Free branch protection to defend against COP/JOP.

Both capabilities introduce new instruction set extensions, and are described in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D.

Control-Flow Enforcement Technology introduces a new exception (#CP) with interrupt vector 21.

18.1.1 Shadow Stack

A shadow stack is a second stack for the program that is used exclusively for control transfer operations. This stack is separate from the data stack and can be enabled for operation individually in user mode or supervisor mode. When shadow stacks are enabled, the CALL instruction pushes the return address on both the data and shadow stack. The RET instruction pops the return address from both stacks and compares them. If the return addresses from the two stacks do not match, the processor signals a control protection exception (#CP). Note that the shadow stack only holds the return addresses and not parameters passed to the call instruction.

The shadow stack is protected from tamper through the page table protections such that regular store instructions cannot modify the contents of the shadow stack. To provide this protection the page table protections are extended to support an additional attribute for pages to mark them as "Shadow Stack" pages. When shadow stacks are enabled, control transfer instructions/flows like near call, far call, call to interrupt/exception handlers, etc. store return addresses to the shadow stack and the access will fault if the underlying page is not marked as a "Shadow Stack" page. However stores from instructions like MOV, XSAVE, etc. will not be allowed. Likewise control transfer instructions like near RET, far RET, IRET, etc. when they attempt to read from the shadow stack the access will fault if the underlying page is not marked as a "Shadow Stack" page. This paging protection detects and prevents conditions that cause an overflow or underflow of the shadow stack when the shadow stack is delimited by non-shadow stack guard pages, or any malicious attempts to redirect the processor to consume data from addresses that are not shadow stack addresses.

18.1.2 Indirect Branch Tracking

The ENDBRANCH instruction is a new instruction that is used to mark valid jump target addresses of indirect calls and jumps in the program. This instruction opcode is selected to be one that is a NOP on legacy machines such that programs compiled with ENDBRANCH new instruction continue to function on old machines without the CET enforcement. On processors that support CET the ENDBRANCH is still a NOP and is primarily used as a marker instruction by the processor pipeline to detect control flow violations. The CPU implements a state machine that tracks indirect JMP and CALL instructions. When one of these instructions is executed, the state machine moves from IDLE to WAIT_FOR_ENDBRANCH state. In WAIT_FOR_ENDBRANCH state the next instruction in the program

stream must be an ENDBRANCH. If the next instruction is not an ENDBRANCH, the processor causes a control protection exception (#CP); otherwise, the state machine moves back to IDLE state.

18.1.3 Speculative Behavior when CET is Enabled

Speculative execution of near indirect JMP/CALL/RET indirect branches may be able to create an active side channel vulnerability that reveals the contents of data.

There are two basic methods that an attacker may be able to use to control indirect branch speculation in order to speculatively execute code that causes a side channel:

1. Attacker controlled prediction.
2. Attacker controlled jump redirection.

With attacker controlled prediction, the attacker trains indirect branch predictors such that the desired victim indirect branch goes to the attacker desired location. Examples include Branch Target Injection (also called "Variant 2" and "Spectre") and RSB wrap on underflow (also called "ret2spec").

With attacker controlled jump redirection, the attacker controls a speculative-only value used as input to the indirect branch so that the branch mispredicts to the attacker desired location. Examples of this include Bound Check Bypass Store (where a speculative store containing an attacker controlled value may overwrite the indirect branch target before the load of the target) and Speculative Store Bypass (where a load of the indirect branch target may bypass the most recent store of the target value and thus speculatively read an older attacker controlled value at the same memory location).

In addition to the existing mitigation features like IBRS, STIBP, and IBPB, processors supporting CET will have a variety of additional features to constrain control flow speculation in order to mitigate such attacks. For details on these features, see Section 18.2.6, "Constraining Execution at Targets of RET," and Section 18.3.8, "Constraining Speculation after Missing ENDBRANCH."

18.2 SHADOW STACKS

A shadow stack is a second expand down stack used exclusively for control transfer operations. This stack is separate from the data stack. The shadow stack is not used to store data and hence is not explicitly writeable by software. Writes to the shadow stack are restricted to control transfer instructions and shadow stack management instructions. The shadow stack feature can be enabled separately in user mode (CPL == 3) or supervisor mode (CPL < 3).

Shadow stacks operate only in protected mode. Shadow stacks cannot be enabled in virtual 8086 mode.

It is recommended to not configure the shadow stack in the linear address range 0 to 64 KB or adjacent to the canonical address boundary.

18.2.1 Shadow Stack Pointer and its Operand and Address Size Attributes

When CET is enabled the processor supports a new architectural register, shadow stack pointer (SSP), when the processor supports the shadow stack feature. The SSP cannot be directly encoded as a source, destination or memory operand in instructions. The SSP points to the current top of the shadow stack.

The width of the shadow stack is 32-bit in 32-bit/compatibility mode and is 64-bit in 64-bit mode. The address-size attribute of the shadow stack is likewise 32-bit in 32-bit/compatibility mode and 64-bit in 64-bit mode.

18.2.2 Terminology

When shadow stacks are enabled, certain control transfer instructions/flows and shadow stack management instructions do loads and stores from and to the shadow stack. Such loads and stores from control transfer instructions and shadow stack management instructions are termed as **shadow-stack loads** and **shadow-stack stores** to distinguish them from a loads and stores performed by other instructions like MOV, XSAVES, etc.

The pseudocode for the instruction operations use the notation `ShadowStackEnabled(CPL)` as a test of whether shadow stacks are enabled at the CPL. This term returns a TRUE or FALSE indication as follows.

```
ShadowStackEnabled(CPL):
  IF CR4.CET = 1 AND CR0.PE = 1 AND EFLAGS.VM = 0
    IF CPL = 3
      THEN
        (* Obtain the shadow stack enable from IA32_U_CET MSR (MSR address 6A0H) used to enable
           feature for CPL = 3 *)
        SHADOW_STACK_ENABLED = IA32_U_CET.SH_STK_EN;
      ELSE
        (* Obtain the shadow stack enable from IA32_S_CET MSR (MSR address 6A2H) used to enable
           feature for CPL < 3 *)
        SHADOW_STACK_ENABLED = IA32_S_CET.SH_STK_EN;
    FI;
    IF SHADOW_STACK_ENABLED = 1
      THEN
        return TRUE;
      ELSE
        return FALSE;
    FI;
  ELSE
    (* Shadow stacks not enabled in real mode and virtual-8086 mode or if the master CET feature
       enable in CR4 is disabled *)
    return FALSE;
  ENDIF
```

Additionally, the following terms are used.

- `ShadowStackPush4B`: Decrements the shadow stack pointer (SSP) by 4 bytes and copies the 4 byte source operand to the top of the shadow stack.
- `ShadowStackPush8B`: Decrements the shadow stack pointer (SSP) by 8 bytes and copies the 8 byte source operand to the top of the shadow stack.
- `ShadowStackPop4B`: Copies 4 bytes at the current top of stack (indicated by the SSP register) to the location specified with the destination operand. It then increments the SSP register by 4 bytes to point to the new top of stack.
- `ShadowStackPop8B`: Copies 8 bytes at the current top of stack (indicated by the SSP register) to the location specified with the destination operand. It then increments the SSP register by 8 bytes to point to the new top of stack.
- `shadow_stack_lock_cmpxchg8B(address, new_value, expected_value)`: this function executes atomically and compares the `expected_value` to the 8 byte read from memory specified by the address operand using a locked shadow-stack load. If the two values are equal, the `new_value` is written to the address using an unlocking shadow-stack store. If the two values are not equal, then the value read by the shadow-stack load is written back, also using an unlocking shadow-stack store. The function returns the value read from the memory specified by the address operand.

18.2.3 Supervisor Shadow Stack Token

This section describes shadow-stack support that is provided for call gates, IDT event delivery, and IRET. This behavior does not apply when FRED transitions are enabled.

On an inter-privilege far CALL or when calling an interrupt/exception handler at a higher privilege level, a stack switch occurs; if shadow stacks are enabled at the new privilege level, then a shadow stack switch occurs. Shadow stacks that can be switched to by hardware as part of a privilege change are required to have a supervisor shadow stack token set up by the supervisor to provide the address of the new SSP register. The supervisor shadow stack tokens also serve the purpose of enforcing that a shadow stack can be made active on only one logical processor

when switched to by the processor. The supervisor shadow stack token must be set up only on shadow stacks intended to be used on these transfers. The address of the supervisor shadow stack token is programmed into the `IA32_PLx_SSP` MSR (where $0 \leq x \leq 2$). The `WRMSR` and `XRSTORS` instructions require the address specified in the `IA32_PLx_SSP` MSR (where $0 \leq x \leq 2$) to be 4 byte aligned; otherwise, the instruction causes a general protection exception (`#GP(0)`).

The supervisor shadow stack token is a 64-bit value formulated as follows.

- Bit 63:3: Bits 63:3 of the linear address of the supervisor shadow stack token.
- Bit 2: Reserved. Must be zero.
- Bit 1: Reserved. Must be zero.
- Bit 0: Busy bit. If 0, indicates this shadow stack is not active on any logical processor. If 1, indicates this shadow stack is currently active on one of the logical processors.

The following figure illustrates a supervisor shadow stack with a supervisor shadow stack token located at its base.

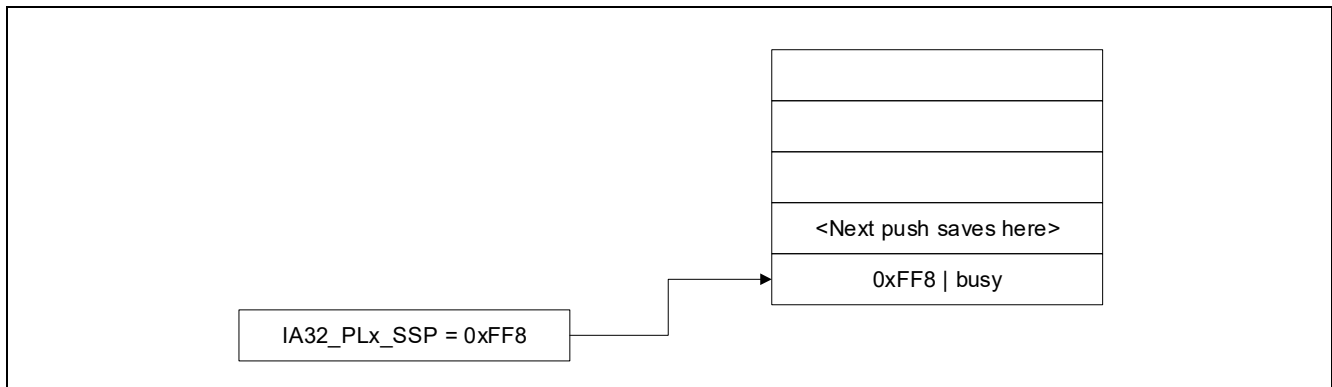


Figure 18-1. Supervisor Shadow Stack with a Supervisor Shadow Stack Token

If the far `CALL` or event delivery will push a 24-byte stack frame after the token is acquired, the 8-byte supervisor shadow stack token and the stack frame must be fully contained within a 32-byte region that is aligned to 32-bytes on the shadow stack. If they are not, a general-protection exception (`#GP(0)`) occurs.

The processor does the following checks prior to switching to a supervisor shadow stack programmed into the `IA32_PLx_SSP` MSR. These steps are performed atomically.

1. Load the supervisor shadow stack token from the address specified in the `IA32_PLx_SSP` MSR using a locked shadow-stack store.
2. Check if the busy bit in the token is 0; reserved bits must be 0.
3. Check if the address programmed in the MSR matches the address in the supervisor shadow stack token; reserved bits must be 0.
4. If checks 2 and 3 are successful, then set the busy bit in the token using an unlocking shadow-stack store and switching the SSP to the value specified in the `IA32_PLx_SSP` MSR.
5. If checks 2 or 3 fail, write back the value read at step 1 using an unlocking shadow-stack store (the busy bit is not set) and raise a `#GP(0)` exception.

If the far `CALL` or event delivery pushes a stack frame after the token is acquired and any of the pushes causes a fault or VM exit, the processor will revert to the old shadow stack and the busy bit in the new shadow stack's token remains set. The new shadow stack is said to be **prematurely busy**. Software should enable supervisor shadow stacks only if it is certain that this situation cannot occur. If `CPUID.07H.01H:EDX[18]` is enumerated as 1, it is sufficient for an operating system to ensure that none of the pushes can cause a page fault.

On a far `RET` to a lesser privilege level or on an `IRET` that switches shadow stack, the instruction clears the busy bit in the shadow stack token as follows. These steps are also performed atomically.

1. Load the supervisor shadow stack token from the SSP using a locked shadow-stack load.

2. Check if the busy bit in the token is 1; reserved bits must be 0.
3. Check if the address programmed in supervisor shadow stack token matches SSP; reserved bits must be 0.
4. If checks 2 and 3 are successful, then write back the token with an unlocking shadow-stack store, clearing the busy bit; otherwise, write back the value read at step 1 using an unlocking shadow-stack store and continue without modifying the contents of the shadow stack pointed to by SSP.

18.2.4 Shadow Stack Usage on Task Switch

A task switch (see Chapter 10, “Task Management,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A) may be invoked by:

- JMP or CALL instruction to a TSS descriptor in the GDT.
- JMP or CALL instruction to a task-gate descriptor in the GDT or the current LDT.
- An interrupt or exception vector points to a task-gate descriptor in the IDT.

With shadow stack enabled, the new task must be associated with a 32-bit TSS and must not be in virtual-8086 mode. The 32-bit SSP for the new task is located at offset 104 in the 32-bit TSS. Thus the TSS of the new task must be at least 108 bytes. This SSP is required to be 8 byte aligned, and required to point to a “supervisor shadow stack” token (though the task may be at CPL3).

On a task switch initiated by a CALL instruction, an interrupt, or exception, the SSP of the old task is pushed onto the shadow stack of the new task along with the CS and LIP of the old task. This is true even for a nested task switch initiated by a CALL instruction. Likewise, on a task switch initiated by IRET, the SSP of the new task is restored from the shadow stack of old task. The CS and LIP on the shadow stack of the old task are matched against the return address determined by the CS and EIP of the new task. If the match fails, a #CP(FAR-RET/IRET) exception is reported.

18.2.5 Switching Shadow Stacks

The architecture provides a mechanism to switch shadow stacks using a pair of instructions; RSTORSSP and SAVEPREVSSP. The RSTORSSP instruction verifies a shadow-stack-restore token located at the top of the new shadow stack and referenced by the memory operand of this instruction. After RSTORSSP determines the validity of the restore point on the new shadow stack, it switches the SSP to point to the token. The shadow-stack-restore token is a 64-bit value formatted as follows.

- Bit 63:2: Value of shadow stack pointer when this restore point was created.
- Bit 1: Reserved. Must be zero.
- Bit 0: Mode bit. If 0, the token is a compatibility/legacy mode shadow-stack-restore token. If 1, then this shadow stack restore token can be used with a RSTORSSP instruction in 64-bit mode.

The shadow-stack-restore token is created by the SAVEPREVSSP instruction. The operating system may also create a restore point on a shadow stack by creating a shadow-stack-restore token.

Once the shadow stack has been switched to a new shadow stack by the RSTORSSP instruction, software can create a restore point on the old shadow stack by executing the SAVEPREVSSP instruction. In order to allow the SAVEPREVSSP instruction to determine the address where to save the shadow-stack-restore token, the RSTORSSP instruction replaces the shadow-stack-restore token with a previous-ssp token that holds the value of the SSP at the time the RSTORSSP instruction was invoked. The previous-ssp token is formatted as follows.

- Bit 63:2: Shadow stack pointer when the RSTORSSP instruction was invoked, i.e., the SSP of the old shadow stack.
- Bit 1: Set to 1.
- Bit 0: Mode bit. If 0, then this previous-ssp token can be used with a SAVEPREVSSP instruction in compatibility/legacy mode. If 1, then this previous-ssp token can be used with a SAVEPREVSSP instruction in 64-bit mode.

The following figure illustrates the RSTORSSP instruction operation during a shadow stack switching sequence.

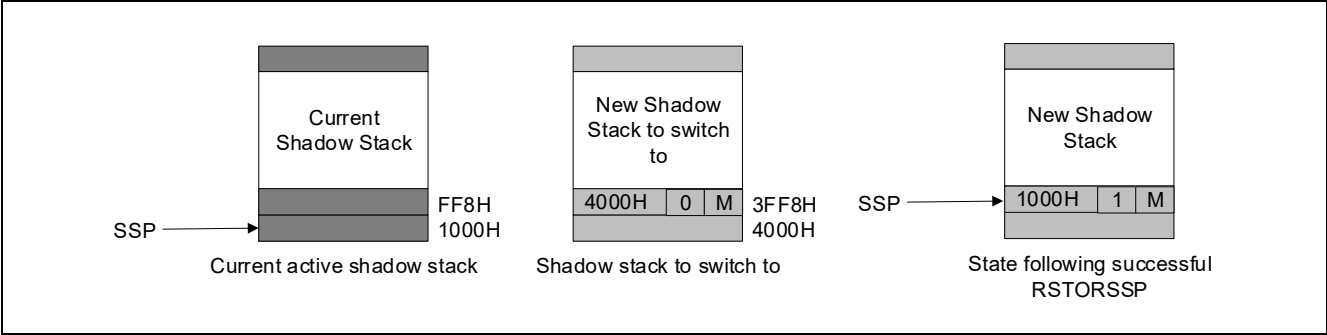


Figure 18-2. RSTORSSP to Switch to New Shadow Stack

In this example, the initial SSP is 1000H and the shadow-stack-restore token is on a new shadow stack at address 3FF8H. The token at address 3FF8H holds the SSP when this restore point was created; in this example it is 4000H.

In order to switch to the new shadow stack, the RSTORSSP instruction is invoked with the memory operand pointing set to 3FF8H. When the RSTORSSP instruction completes, the SSP is set to 3FF8H and the shadow-stack-restore token at 3FF8H is replaced by a previous-ssp token that holds the address 1000H, i.e., the old SSP.

The following figure illustrates the SAVEPREVSSP instruction operation during a shadow stack switching sequence.

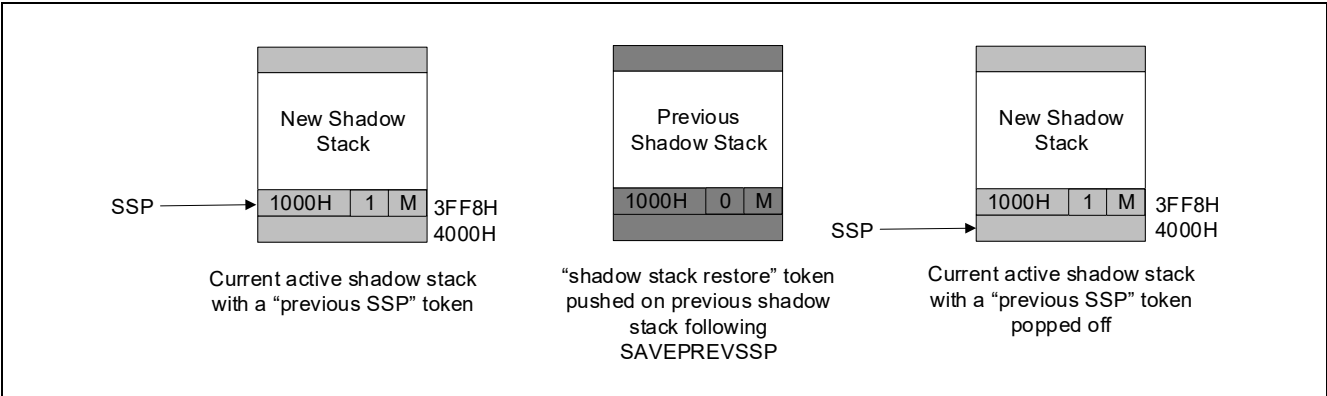


Figure 18-3. SAVEPREVSSP to Save a Restore Point

To allow switching back to this old shadow stack, a SAVEPREVSSP instruction is now invoked. The SAVEPREVSSP instruction does not take any memory operand and expects to find a previous-ssp token at the top of the shadow stack, i.e., at address 3FF8H. The SAVEPREVSSP instruction then saves a shadow-stack-restore token on the old shadow stack at address FF8H, and the token itself holds the address 1000H which is the address recorded in the previous-ssp token. The SAVEPREVSSP instruction also pops the previous-ssp token off the current shadow stack and thus the SSP following SAVEPREVSSP is 4000H.

Subsequently to switch back to the old shadow stack, a RSTORSSP instruction may be invoked with memory operand set to FF8H.

If, following a switch to a new shadow stack, it is not required to create a restore point on the old shadow stack, then the previous-ssp token created by the RSTORSSP instruction can be popped off the shadow stack by using the INCSSP instruction.

See the SAVEPREVSSP and RSTORSSP instruction operations for the detailed algorithm.

18.2.6 Constraining Execution at Targets of RET

Instructions at the target of a RET instruction will not execute, even speculatively, if the RET addresses (either from normal stack or shadow stack) are speculative-only or do not match, unless the target of the RET is also predicted (e.g., by a Return Stack Buffer prediction), when CET shadow stack is enabled. A RET address would be speculative-only if it was modified by an older speculative-only store, or was an older value than the most recent value stored to that address on the logical processor.

18.3 INDIRECT BRANCH TRACKING

When the indirect branch tracking feature is active, the indirect JMP/CALL instruction behavior changes as follows.

- **JMP:** If the next instruction retired after an indirect JMP is not an ENDBR32 instruction in legacy and compatibility mode, or ENDBR64 instruction in 64-bit mode, then a #CP fault is generated. Below JMP instructions are tracked to enforce an ENDBRANCH. Note that Jcc, RIP relative, and far direct JMP are not included as these have an offset encoded into the instruction and are not exploitable to create unintended control transfers.
 - JMP r/m16, JMP r/m32, JMP r/m64
 - JMP m16:16, JMP m16:32, JMP m16:64
- **CALL:** If the next instruction retired after an indirect CALL is not an ENDBR32 instruction in legacy and compatibility mode, or ENDBR64 in 64-bit mode, then a #CP fault is generated. Below CALL instructions are tracked to enforce an ENDBRANCH. Note that relative and zero displacement forms of CALL instructions are not included as these have an offset encoded into the instruction and are not exploitable to create unintended control transfers.
 - CALL r/m16, CALL r/m32, CALL r/m64
 - CALL m16:16, CALL m16:32, CALL m16:64

The ENDBR32 and ENDBR64 instructions will have the same effect as the NOP instruction on Intel 64 processors that do not support CET. On processors supporting CET, these instructions do not change register or flag state. This allows CET instrumented programs to execute on processors that do not support CET. Even when CET is supported and enabled, these NOP-like instructions do not affect the execution state of the program, do not cause any additional register pressure, and are minimally intrusive from power and performance perspectives.

The processor implements two dual-state machines to track indirect CALL/JMP for terminations. One state machine is maintained for user mode and one for supervisor mode. At reset the user and supervisor mode state machines are in IDLE state.

On instructions other than indirect CALL/JMP, the state machine stays in the IDLE state.

On an indirect CALL or JMP instruction, the state machine transitions to the WAIT_FOR_ENDBRANCH state.

In the WAIT_FOR_ENDBRANCH state, the indirect branch tracking state machine verifies the next instruction is an ENDBR32 instruction in legacy and compatibility mode, or ENDBR64 instruction in 64-bit mode, and either:

- Causes a #CP fault, or
- Allows the next instruction if legacy compatibility configuration allows (see Section 18.3.6).

The priority of the #CP(ENDBRANCH) exception relative to other events is as follows.

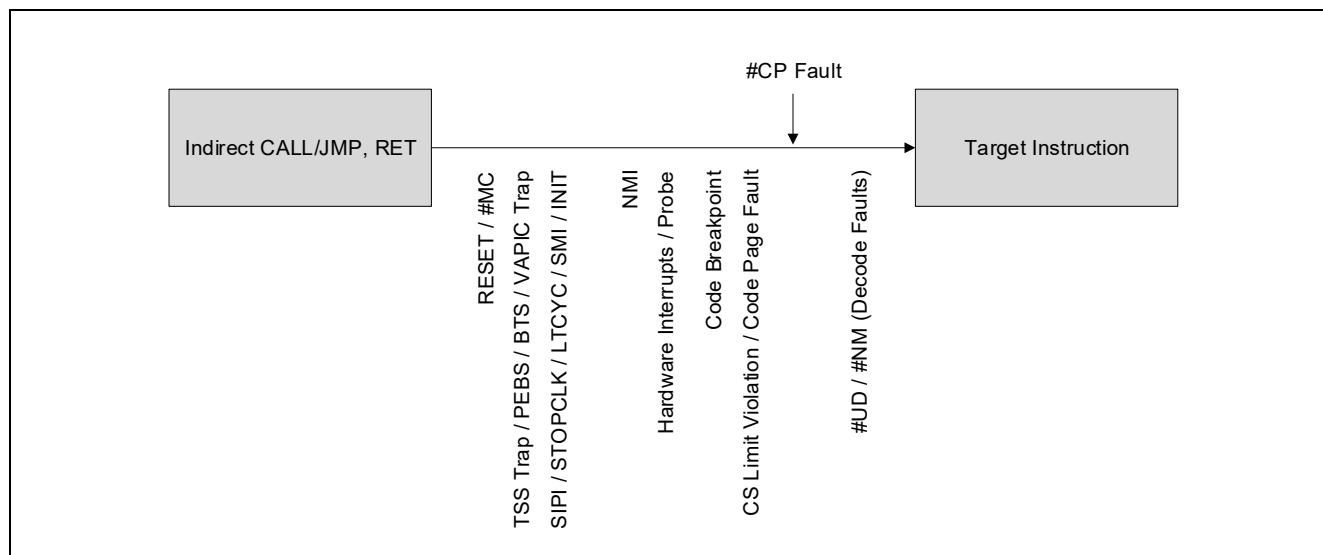


Figure 18-4. Priority of Control Protection Exception on Missing ENDBRANCH

Higher priority faults/traps/events that occur at the end of an indirect CALL/JMP are delivered ahead of any #CP(ENDBRANCH) fault. The CET state machine at the privilege level where the higher priority fault/trap/event occurred retains its state when the control transfers to the fault/trap/event handler. The instruction pointer pushed on the stack for a #CP(ENDBRANCH) fault is the address of the instruction at the target of the indirect CALL/JMP that caused the fault.

18.3.1 No-track Prefix for Near Indirect CALL/JMP

CET allows software to designate certain indirect CALL and JMP instructions as “non-tracked indirect control transfer instructions”. Software (e.g., compiler generated code for switch statements, jump tables, etc.) should use the no-track prefix only if they have generated code to validate the possible targets of this CALL/JMP to be legal targets. Software (e.g., compilers), when using the no-track prefix with CALL/JMP where an absolute offset is specified indirectly in a memory location, should ensure that such memory locations cannot be tampered. When enabled by setting the NO_TRACK_EN control in the IA32_U_CET/IA32_S_CET MSR, near indirect CALL and JMP instructions when prefixed with 3EH do not modify the CET indirect branch tracker. Far CALL and JMP instructions are always tracked and ignore the 3EH prefix. When this control is 0, near indirect CALL and JMP instructions are always tracked irrespective of the presence of the 3EH prefix.

In 64-bit mode, the 3EH prefix on an indirect CALL or JMP is recognized as a no-track prefix if there isn’t a 64H/65H prefix on the instruction.

In legacy/compatibility mode, the 3EH prefix on an indirect CALL or JMP is recognized as a no-track prefix when it is the last group 2 prefix on the instruction.

18.3.2 Terminology

The pseudocode for the instruction operations use a notation `EndbranchEnabled(CPL)` as a test of whether indirect branch tracking is enabled at the CPL. This term returns a TRUE or FALSE indication as follows.

`EndbranchEnabled(CPL);`

```

    IF CR4.CET = 1 AND CR0.PE = 1 AND EFLAGS.VM = 0
        IF CPL = 3
            THEN
                (* Obtain the ENDBRANCH enable from MSR used to enable feature for CPL = 3 *)
                ENDBR_ENABLED = IA32_U_CET.ENDBR_EN;
            ELSE
                (* Obtain the ENDBRANCH enable from MSR used to enable feature for CPL < 3 *)
                ENDBR_ENABLED = IA32_S_CET.ENDBR_EN;
        FI;
        IF ENDBR_ENABLED = 1
            THEN
                return TRUE;
            ELSE
                return FALSE;
        FI;
    ELSE
        (* Indirect branch tracking is not enabled in real mode and virtual-8086 mode or if the master CET feature
        enable in CR4 is disabled *)
        return FALSE;
    ENDIF

```

Likewise the notation `EndbranchEnabledAndNotSuppressed` is defined as follows:

`EndbranchEnabledAndNotSuppressed(CPL);`

```

    IF CR4.CET = 1 AND CR0.PE = 1 AND EFLAGS.VM = 0
        IF CPL = 3
            THEN
                (* Obtain the ENDBRANCH enable from MSR used to enable feature for CPL = 3 *)
                ENDBR_ENABLED = IA32_U_CET.ENDBR_EN;
                SUPPRESSED = IA32_U_CET.SUPPRESS;
            ELSE
                (* Obtain the ENDBRANCH enable from MSR used to enable feature for CPL < 3 *)
                ENDBR_ENABLED = IA32_S_CET.ENDBR_EN;
                SUPPRESSED = IA32_S_CET.SUPPRESS;
        FI;
        IF ENDBR_ENABLED = 1 AND SUPPRESSED = 0
            THEN
                return TRUE;
            ELSE
                return FALSE;
        FI;
    ELSE
        (* Indirect branch tracking is not enabled in real mode and virtual-8086 mode or if the master CET feature
        enable in CR4 is disabled *)
        return FALSE;
    ENDIF

```

18.3.3 Indirect Branch Tracking

The hardware implements two CET indirect branch tracker state machines, one for user mode (CPL == 3) and one for supervisor mode (CPL < 3). At any time, which of the CET indirect branch trackers is in the active state depends on the CPL of the machine. When a user space program is executing, the CPL 3 CET indirect branch tracker is active. When supervisor mode software is executing, the CPL < 3 tracker is active. This section describes the various control transfer conditions and the tracker state on those transfers.

18.3.3.1 Control Transfers between CPL 3 and CPL < 3

Some events and instructions can cause control transfer to occur from CPL 3 to CPL < 3, and vice versa. As part of the CPL change the hardware also switches the active CET indirect branch tracker. For example, when an interrupt occurs during execution of a user mode (CPL == 3) program and it causes the CPL to switch to supervisor mode (CPL < 3) then, as part of the CPL change, the user mode CET indirect branch tracker becomes inactive and the supervisor mode CET indirect branch tracker becomes active. A subsequent IRET is used by the interrupt handler to return to the interrupted user mode program. This IRET causes the processor to switch the CPL to user mode (CPL == 3) and, as part of the CPL change, the supervisor mode CET indirect branch tracker becomes inactive and the user mode CET indirect branch tracker becomes active.

The CPL where the event or instruction that caused the control transfer occurs is termed the source CPL, and the CET indirect branch tracker state at that CPL is referred here as the source CET indirect branch tracker state. The CPL reached at the end of the control transfer is termed the destination CPL, and the CET indirect branch tracker state at that CPL is referred to as the destination CET indirect branch tracker state.

This section describes various cases of control transfers that occur between user mode (CPL 3) and supervisor mode (CPL < 3).

In all these cases the source CET indirect branch tracker state becomes not active and retains its state (IDLE, WAIT_FOR_ENDBRANCH), and the target CET indirect branch tracker state becomes active if there was no fault during the transfer.

- Case 1: Far CALL/JMP, SYSCALL/SYSENTER
 - If indirect branch tracking is enabled, the target indirect branch tracker state becomes active and is unsuppressed and goes to WAIT_FOR_ENDBRANCH. This enforces that the subroutine invoked by a far CALL/JMP must begin with an ENDBRANCH.
- Case 2: Hardware interrupt/trap/exception/NMI/Software interrupt/Machine Checks
 - If indirect branch tracking is enabled, the target indirect branch tracker state becomes active and is unsuppressed and goes to WAIT_FOR_ENDBRANCH.
- Case 3: IRET/Far RET
 - If indirect branch tracking enabled, the target indirect branch tracker becomes active and keeps its state. If the user mode was interrupted by a higher priority event, like an interrupt at the end of the indirect CALL/JMP, then when an IRET or Far RET is used to return to the interrupted user mode program, the user mode indirect branch tracker retains its state and a #CP fault will occur if the next instruction decoded is not an ENDBR32/64 according to mode of machine.

18.3.3.2 Control Transfers within CPL 3 or CPL < 3

Some events and instructions can cause control transfer to occur within CPL 3 or CPL < 3. For such transfers since the CPL class does not change, the same indirect branch tracker is used at the beginning and end of the control transfer.

- Case 1: Far CALL/JMP, Near indirect CALL/JMPCALL/JMP
 - Far CALL/JMP: If indirect branch tracking is enabled, active indirect branch tracker is unsuppressed and goes to WAIT_FOR_ENDBRANCH.
 - Near indirect CALL/JMPCALL/JMP: If indirect branch tracking is enabled and not suppressed, active indirect branch tracker goes to WAIT_FOR_ENDBRANCH.
- Case 2: Hardware interrupt/trap/exception/NMI/Software interrupt/Machine Checks

- If indirect branch tracking is enabled, the active indirect branch tracker is unsuppressed and goes to WAIT_FOR_ENDBRANCH.
- Case 3: IRET
 - If indirect branch tracking is enabled, the active indirect branch tracker keeps its state.

18.3.4 Indirect Branch Tracking State Machine

The state machine is described by Table 18-1.

Table 18-1. Indirect Branch Tracking State Machine

Current State	Trigger	Next State
TRACKER=IDLE, SUPPRESS=0, ENDBR_EN=1	Instructions other than indirect CALL/JMP or 3EH prefixed near indirect CALL/JMP and NO_TRACK_EN=1	TRACKER=IDLE, SUPPRESS=0, ENDBR_EN=1
	Indirect CALL/JMP without 3EH prefix Indirect CALL/JMP with 3EH prefix and NO_TRACK_EN=0 Far CALL/JMP	TRACKER=WAIT_FOR_ENDBRANCH, SUPPRESS=0, ENDBR_EN=1
TRACKER= WAIT_FOR_ENDBRANCH, SUPPRESS=0, ENDBR_EN=1	INT3/INT1	TRACKER= WAIT_FOR_ENDBRANCH, SUPPRESS=0, ENDBR_EN=1
	ENDBRANCH instruction	TRACKER=IDLE, SUPPRESS=0, ENDBR_EN=1
	Successful ENCLU[ERESUME]	TRACKER=IDLE, SUPPRESS=0, ENDBR_EN=1
	Instructions other than ENDBRANCH, successful ENCLU[ERESUME] or INT3 or INT1	If legacy compatibility treatment is not enabled or if not allowed by legacy code page bitmap: ▪ No state change and deliver #CP (ENDBRANCH) If legacy compatibility treatment is enabled and transfer allowed by legacy code page bitmap: ▪ TRACKER=IDLE, SUPPRESS=ISUPPRESS_DIS, ENDBR_EN=1
TRACKER=x, SUPPRESS=x, ENDBR_EN=0	All instructions	TRACKER=x, SUPPRESS=x, ENDBR_EN=0
TRACKER=IDLE, SUPPRESS=1, ENDBR_EN=1	Far CALL/JMP, INTn/INT3/INT0	TRACKER=WAIT_FOR_ENDBRANCH, SUPPRESS=0, ENDBR_EN=1
	ENDBRANCH instruction Successful ENCLU[ERESUME]	TRACKER=IDLE, SUPPRESS=0, ENDBR_EN=1
	All other instructions including indirect CALL/JMP	TRACKER=IDLE, SUPPRESS=1, ENDBR_EN=1
TRACKER=1, SUPPRESS=1, ENDBR_EN=1 (This state cannot be reached by hardware and is disallowed as a valid state by WRMSR/XRSTORS/VM entry/VM exit)	NA	NA

18.3.5 INT3 Treatment

INT3 are treated special in the WAIT_FOR_ENDBRANCH state. Occurrence of INT3 do not move the tracker to IDLE but instead the #BP trap from the INT3 instructions respectively is delivered as a higher priority event than the #CP exception due to missing ENDBRANCH.

Inside an enclave, INT3 delivers a fault-class exception and thus does not require the CPL to be less than DPL in the IDT gate 3. Following opt-out entry, the instruction delivers #UD. Following opt-in entry, INT3 delivers #BP. The special treatment of INT3 in WAIT_FOR_ENDBRANCH state does not apply in enclave mode following opt-out entry.

18.3.6 Legacy Compatibility Treatment

ENDBRANCH legacy compatibility treatment allows a CET enabled program to be used with legacy software that was not compiled / instrumented with ENDBRANCH. A CET enabled program enters legacy compatibility treatment when all of the below conditions are met.

1. Legacy compatibility configuration is enabled in this CPL class by setting the LEG_IW_EN bit in IA32_U_CET/IA32_S_CET.
2. Control transfer is performed using an indirect CALL/JMP without no-track prefix to an instruction other than ENDBRANCH.
3. The legacy code page bitmap is setup to indicate that the target of the control transfer is a legacy code page.

The legacy code page bitmap is a data structure in program memory that is used by the hardware to determine if the code page to which a legacy transfer is being performed is allowed. The access rights for accessing the legacy code page bitmap is determined by the current privilege level (CPL). The legacy code page bitmap is expected to be setup as a read-only data structure.

When a matching ENDBRANCH instruction is not decoded at the target of an indirect CALL/JMP when required, the processor performs the below actions.

CET indirect branch tracking state machine violation event handler:

If LEG_IW_EN == 1

LA = LIP;

IF ENCLAVE_MODE == 1

LA = LA - SECS.BASEADDR;

ENDIF

(* Load byte from bitmap. Address-size attribute for this load is 64 bits if IA32_EFER.LMA is 1 and is 32 bits when IA32_EFER.LMA is 0 *)

IF (IA32_EFER.LMA & CS.L) == 0

BITMAP_BYTE = load 1 byte from address (BITMAP_BASE + LA[31:15])

ELSE IF (CR4.LA57 == 0)

BITMAP_BYTE = load 1 byte from address (BITMAP_BASE + LA[47:15])

ELSE

BITMAP_BYTE = load 1 byte from address (BITMAP_BASE + LA[56:15])

FI;

IF BITMAP_BYTE & (1 << LA[14:12]) == 0 then Deliver #CP(ENDBRANCH) fault

IF CPL = 3

IA32_U_CET.TRACKER = IDLE

IA32_U_CET.SUPPRESS = IA32_U_CET.SUPPRESS_DIS == 0 ? 1 : 0

ELSE

IA32_S_CET.TRACKER = IDLE

IA32_S_CET.SUPPRESS = IA32_S_CET.SUPPRESS_DIS == 0 ? 1 : 0

ENDIF

Restart the instruction (handle all arch. consistency around MOV SS state machines, STI etc.) without opening up interrupt/trap window.

ELSE

Deliver #CP(ENDBRANCH) Fault

ENDIF

Faults/traps in pseudocode are delivered normally (e.g., #PF, EPT violation). On a fault, the active tracker holds the last value (WAIT_FOR_ENDBRANCH) and the address saved on the stack is the current IP (instruction that wasn't the ENDBRANCH).

The CET indirect branch tracking state machine is suppressed in legacy compatibility mode if the SUPPRESS_DIS control bit is 0.

Once the CET indirect branch tracking state machine has been suppressed, subsequent indirect CALL/JMP are not tracked for termination instruction.

Once CET indirect branch tracking has been suppressed, subsequent execution of ENDBRANCH instructions will do the following (see the ENDBR32 and ENDBR64 instructions in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A for details).

```
IF EndbranchEnabled(CPL) == 0
```

```
    NOP
```

```
ELSE
```

```
    SUPPRESS = 0
```

```
    TRACKER = IDLE
```

```
ENDIF
```

18.3.6.1 Legacy Code Page Bitmap Format

The legacy code page bitmap is a flat bitmap whose linear address is pointed to by the EB_LEG_BITMAP_BASE. Each bit in the bitmap represents a 4K page in linear memory. If the bit is 1 it indicates that the corresponding code page is a legacy code page; else it is a CET-enabled code page.

The processor uses the linear address of the instruction to which legacy transfer was attempted to lookup the bitmap. Bits of the linear address used as index in the bitmap are as follows.

- In legacy and compatibility mode: Bits 31:12.
- In 64-bit mode (EFER.LMA=1 and CS.L=1): Bits 47:12.

18.3.7 Other Considerations

18.3.7.1 Intel® Transactional Synchronization Extensions (Intel® TSX) Interactions

The XBEGIN instruction encodes the relative offset to the abort handler and hence the fallback to the abort handler can be considered as a "direct" branch and the abort handler does not need to have an ENDBRANCH.

CET continues to enforce indirect CALL/JMP tracking within a transaction. Legacy compatibility treatment inside a transaction functions normally. If a transaction abort occurs then the processor sets the state of the indirect branch tracker to IDLE and not-suppressed.

18.3.7.2 #CP(ENDBRANCH) Priority w.r.t #NM and #UD

#NM, #UD and #CP(ENDBRANCH) are opcode based faults. However, #CP(ENDBRANCH) is in a higher priority class than #NM and #UD as CET architecturally requires an ENDBRANCH at target of indirect CALL/JMP.

18.3.7.3 #CP(ENDBRANCH) Priority w.r.t #BP and #DB

Debug Exceptions priority is as follows.

- Traps delivered before any #CP(ENDBRANCH) fault: Data breakpoint trap, IO breakpoint trap single step trap, task switch trap.
- Code Breakpoint fault detected before instruction decode and delivered before #CP(ENDBRANCH).
- General-detect (GD) exception condition fault: Lower priority than #CP(ENDBRANCH).

- On IRET back from #DB/#BP, the source indirect branch tracker becomes active if enabled and not suppressed. INT3 does not cause #CP(ENDBRANCH) to support debugger usage of replacing bytes of ENDBRANCH with INT3 to set breakpoints. INT3 at target of a CALL-JMP(indirect) cause #BP(INT3) instead of #CP(ENDBRANCH), #CP(ENDBRANCH) fault is delayed. #BP caused by INT3 treated like other events that are higher priority than CET fault. On IRET back from #BP the source indirect tracker becomes active if enabled and not suppressed.

18.3.8 Constraining Speculation after Missing ENDBRANCH

When the CET tracker is in the WAIT_FOR_ENDBRANCH state, instruction execution will be limited or blocked, even speculatively, if the next instruction is not an ENDBRANCH.

This means that when indirect branch tracking is enabled and not suppressed, the instructions at the target of a near indirect JMP/CALL without the no-track prefix will only speculatively execute if there is an ENDBRANCH at the target. This can constrain both attacker controlled prediction as well as attacker controlled jump redirection attacks on near indirect JMPs/CALLs by reducing the gadgets available to an attacker using these techniques. Early implementations of CET may limit the speculative execution to a small number of instructions (less than 8, with no more than 5 loads) past a missing ENDBRANCH, while later implementations will completely block the speculative execution of instructions after a missing ENDBRANCH.

This mechanism also limits or blocks speculation of the next sequential instructions after an indirect JMP or CALL, presuming the JMP/CALL puts the CET tracker into the WAIT_FOR_ENDBRANCH state and the next sequential instruction is not an ENDBRANCH.

18.4 INTEL® TRUSTED EXECUTION TECHNOLOGY (INTEL® TXT) INTERACTIONS

GETSEC[ENTERACCS] and GETSEC[SENDER] clear CR4.CET, and it is not restored when these instructions complete.

GETSEC[EXITAC] will cause #GP(0) fault if CR4.CET is set.

5. Updates to Chapter 21, Volume 1

Change bars and text show changes to Chapter 21 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

Changes to this chapter:

- The following CPUID leaves and subleaves were added:
 - CPUID.23H.04H -- PEBS Capabilities
 - CPUID.23H.05H -- Arch PEBS GP and Fixed Counters Supported
 - CPUID.27H -- RDT-M Asymmetric
 - CPUID.27H.00H -- Monitoring Enumeration
 - CPUID.27H.01H -- L3 Capability Enumeration
 - CPUID.28H -- RDT-A Asymmetric
 - CPUID.28H.00H -- Main
 - CPUID.28H.01H -- L3
 - CPUID.28H.02H -- L2
 - CPUID.28H.03H -- L3 Resource Bandwidth Enforcement
- The following CPUID leaves and subleaves were updated:
 - CPUID.01H:EBX Feature information. Updated last paragraph for clarity on enumerating APIC_ID.
 - CPUID.01H:ECX Feature information. Updated description of ECX[31].
 - CPUID.01H:EDX Feature information.
 - In EDX[APIC, 9], updated physical address range of memory mapped commands.
 - In EDX[DS, 21], corrected field name and updated reference to Chapter 19 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.
 - In EDX[HTT, 28], updated description for APIC ID.
 - CPUID.02H -- TLB/Cache/Prefetch Information.
 - Clarified new descriptor values to refer enumerations to CPUID.04H and CPUID.18H.
 - Added missing rows in Table 21-12, "Encoding of CPUID Leaf 2 Descriptors," for 00H through 09H.
 - CPUID.07H.01H:EAX Extended Feature Information. Added EAX[FRED, 17] and EAX[LKGS, 18] for flexible return and event delivery.
 - CPUID.07H.01H:EBX Extended Feature Information. Added EBX[PBNKDB, 1] for Total Storage Encryption.
 - CPUID.07H.01H:EDX Extended Feature Information. Added EDX[MWAIT, 23] and EDX[SLSM, 24].
 - CPUID.07H.02H:EDX Extended Feature Information. Corrected CPUID.07H.02H in the description.
 - CPUID.0AH -- Architectural Performance Monitoring.
 - Corrected MAX_LEAF parameter to greater than or equal to 0AH for validity of CPUID.0AH in the description.
 - Added EBX[BACKEND_NA, 8] and EBX[BADSPEC_NA, 9].
 - Corrected reference to Chapter 21, "Performance Monitoring", of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3, below the table.
 - CPUID.0BH -- Extended Topology. Added information for Domain Type ID reporting in CPUID.0BH.00H.ECX[15:8] in the description. Edited which registers return 0 when ECX contains an invalid sub-leaf index.
 - CPUID.0DH.00H -- Processor Extended State Main Sub-Leaf.
 - Updated fields EAX[8] and EAX[16:10].
 - Added a reference to Chapter 13.2, "Enumeration of CPUID Support for XSAVE Instructions and XSAVE-Supported Features," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1 in the description below the Leaf 0DH.00H Processor Extended State table.
 - CPUID.0DH.01H -- Feature and Supervisor State Sub-Leaf.

- Updated fields ECX[7:0], ECX[9], and ECX[18:17].
 - Added a reference to Chapter 13.2, "Enumeration of CPUID Support for XSAVE Instructions and XSAVE-Supported Features," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1 in the description below the Leaf 0DH.00H Processor Extended State table.
- CPUID.0DH.SUB-LEAVES -- Sub-leaves. "CPUID.0DH.ECX > 1" was changed to "CPUID.0DH.SUBLEAVES."
 - CPUID.0FH.00H -- Intel® RDT Monitoring Main Sub-Leaf. Updated field name for EDX[1] to L3_MON.
 - CPUID.0FH.01H -- L3 Cache Intel® Resource Director Technology Monitoring. Updated field names for EAX[9] to IO_RDT_CMT, EAX[10] to IO_RDT_MBM, EDX[1] to MBM_L3_TOTAL, and EDX[2] to MBM_L3_LOCAL.
 - CPUID.10H.00H -- Intel® RDT Allocation Main Sub-Leaf. Added EBX[CBA, 5] and EBX[RESOURCE_PRIORITY, 6].
 - CPUID.10H.01H -- L3 Cache Allocation Technology. Updated field name for EBX[31:0] to CAT_L3_CONTENTION.
 - CPUID.10H.02H -- L2 Cache Allocation Technology. Updated field name for EBX[31:0] to CAT_L2_CONTENTION.
 - CPUID.14H.00H -- Intel® PT Main Sub-Leaf. Added EBX[PTTT, 9].
 - CPUID.14H.01H -- Feature Information Sub-Leaf. Added EAX[TRIGGER_CFG_CNT, 10:8], ECX[ICNT, 0], ECX[TRIGGER_PAUSE, 1], and ECX[TRIGGER_DR_MATCH, 15].
 - CPUID.18H -- Deterministic Address Translation Parameters. In the description, corrected register field used to determine validity of leaf 18H to CPUID.18H.00H:EAX[31:0].
 - CPUID.18H.00H -- Main Sub-Leaf. Updated field name of EDX[4:0] to TYPE.
 - CPUID.1BH -- Output Registers Format for All Sub-Leaves. Updated the description of EAX[11:0], SUB_LEAF_TYPE to 0 (Invalid).
 - CPUID.1CH -- Last Branch Records (LBR) Information. Updated the description for ECX[19:16], EVENT_LOGGING_BITMAP.
 - CPUID.1EH.00H -- TMUL Information Main Leaf. Updated EAX[31:0] to reserved.
 - CPUID.1FH -- V2 Extended Topology. Added information for Domain Type ID reporting in CPUID.1FH.00H.ECX[15:8] in the description. Edited which registers return 0 when ECX contains an invalid sub-leaf index.
 - CPUID.21H -- Unimplemented. Updated CPUID21H to unimplemented.
 - CPUID.23H -- Architectural Performance Monitoring Extended. Corrected field name of CPUID.23H.00H:EAX[31:0] in the description.
 - CPUID.24H.00H -- Converged Vector ISA Main Sub-Leaf. Changed widths to lengths in the description for EBX[18:16].
 - CPUID.80000008H -- Extended Function CPUID Information 2. Updated descriptions for EAX[PHYS_ADDR_SIZE, 7:0] and EAX[GUEST_PHYS_ADDR_SIZE, 23:16].

CHAPTER 21

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

When writing software intended to run on Intel processors, it is necessary to identify the type of processor present in a system and the processor features that are available to an application.

The CPUID instruction, known as CPU Identification, was introduced with the Intel® Pentium processor to query the processor's information name space for its identity and supported features. Logically, the CPUID name space comprises a series of nodes indexed by leaf (using the input value of EAX) and in some cases further indexed by sub-leaf (using the input value of ECX). The value of a queried node is returned in EAX, EBX, ECX, and EDX. Note that not all leaves have sub-leaf indexing and the input ECX value will be ignored in those cases. The full description of CPUID can be found in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

All references to "MAX_LEAF" throughout this chapter are used as an abbreviation of "CPUID.00H:EAX.MAX_LEAF".

21.1 IMPORTANT CONSIDERATIONS WHEN USING THE CPUID INSTRUCTION

This section outlines additional factors to consider when using the CPUID instruction.

21.1.1 Guidelines for Using the CPUID Instruction

Use the CPUID instruction for processor identification in the Pentium M processor family, Pentium 4 processor family, Intel Xeon processor family, P6 family, Pentium processor, and later Intel486 processors. This instruction returns the family, model, and (for some processors) a brand string for the processor that executes the instruction. It also indicates the features that are present in the processor and gives information about the processor's caches and TLB.

The ID flag (bit 21) in the EFLAGS register indicates support for the CPUID instruction. If a software procedure can set and clear this flag, the processor executing the procedure supports the CPUID instruction. The CPUID instruction will cause the invalid opcode exception (#UD) if executed on a processor that does not support it.

To obtain processor identification information, a source operand value is placed in the EAX register to select the type of information to be returned. When the CPUID instruction is executed, selected information is returned in the EAX, EBX, ECX, and EDX registers.

The following guidelines are among the most important and should always be followed when using the CPUID instruction to determine available features:

- Always begin by testing for the "GenuineIntel," message in the EBX, EDX, and ECX registers when the CPUID instruction is executed with EAX equal to 0. If the processor is not genuine Intel, the feature identification flags may have different meanings than are described in Intel documentation.
- Test feature identification flags individually and do not make assumptions about undefined bits.

21.1.2 Identification of Earlier Processors

The CPUID instruction is not available in earlier Intel processors up through the earlier Intel 486 processors. For these processors, several other architectural features can be exploited to identify the processor.

The settings of bits 12 and 13 (IOPL), 14 (NT), and 15 (reserved) in the EFLAGS register are different for Intel's 32-bit processors than for the Intel 8086 and Intel 286 processors. By examining the settings of these bits (with the PUSHF/PUSHFD and POPF/POPFD instructions), an application program can determine whether the processor is an 8086, Intel 286, or one of the Intel 32-bit processors:

- 8086 processor — Bits 12 through 15 of the EFLAGS register are always set.
- Intel 286 processor — Bits 12 through 15 are always clear in real-address mode.

- 32-bit processors — In real-address mode, bit 15 is always clear and bits 12 through 14 have the last value loaded into them. In protected mode, bit 15 is always clear, bit 14 has the last value loaded into it, and the IOPL bits depend on the current privilege level (CPL). The IOPL field can be changed only if the CPL is 0.

Other EFLAGS register bits that can be used to differentiate between the 32-bit processors:

- Bit 18 (AC) — Implemented only on the Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors. The inability to set or clear this bit distinguishes an Intel386 processor from the later IA-32 processors.
- Bit 21 (ID) — Determines if the processor is able to execute the CPUID instruction. The ability to set and clear this bit indicates that it is a Pentium 4, Intel Xeon, P6 family, Pentium, or later-version Intel486 processor.

To determine whether an x87 FPU or Numeric Processor Extension (NPX) is present in a system, applications can write to the x87 FPU status and control registers using the FINIT instruction and then verify that the correct values are read back using the FNSTENV instruction.

After determining that an x87 FPU or NPX is present, its type can then be determined. In most cases, the processor type will determine the type of FPU or NPX; however, an Intel386 processor is compatible with either an Intel 287 or Intel 387 math coprocessor.

The method the coprocessor uses to represent ∞ (after the execution of the FINIT, FNINIT, or RESET instruction) indicates which coprocessor is present. The Intel 287 math coprocessor uses the same bit representation for $+\infty$ and $-\infty$; whereas, the Intel 387 math coprocessor uses different representations for $+\infty$ and $-\infty$.

21.1.3 CPUID Basic and Extended Range

The CPUID basic range starts at CPUID.00H and ends at the maximum leaf enumerated in CPUID.00H:EAX.MAX_LEAF[31:0].

The legacy set of CPUID leaves are defined as leaves 00H, 01H, and 02H, which represent the architecture up to and including Pentium II. Processors provided legacy compatibility by limiting the exposed number of leaves to just these legacy leaves by setting IA32_MISC_ENABLE[22] (Limit CPUID Maxval). This is no longer supported on processors that report CPUID.07H.01H:EBX.CPUIDMAXVAL_LIM_RMV[3] as 1; for such processors, IA32_MISC_ENABLE[22] cannot be set to 1 to limit the value returned by CPUID.00H:EAX.MAX_LEAF.

The extended CPUID range starts at leaf 80000000H and ends at the maximum leaf enumerated in CPUID.80000000H:EAX.MAX_EXTENDED_LEAF[31:0].

Older processors before the Pentium 4 do not support the extended CPUID range and treat bit 31 of CPUID's input EAX value as zero.

If a value entered for CPUID.EAX is higher than the maximum input value for basic or extended function for that processor then the data for the highest basic information leaf is returned. Software should not rely on the values returned by the processor outside of the above ranges.

The range CPUID.40000000H to CPUID.4FFFFFFFH do not return feature information for the processor. These are allocated for emulation by software.

21.1.4 CPUID Domains

The fields of each CPUID node are classified into one of several CPUID domains. The fields may be classified separately within a specific node or in aggregate for all the nodes in a leaf or sub-leaf. On a properly configured platform, all logical processors within a CPUID domain return a consistent output value for fields belonging to that domain. As an example, the initial X2APIC ID value returned in CPUID.1FH.00H:EDX[31:0] is classified as being in the Logical Processor Domain because the value is unique for each logical processor in the platform. Whereas, the CLFLUSH Line Size returned in CPUID.00H:EBX[15:8] is classified as Platform Domain because it must be consistent for all logical processor within the entire platform.

- **Platform Domain**—A properly configured platform would provide consistent values for these CPUID fields for each logical processor in the platform.
- **Package Domain**—A properly configured platform provides consistent values for these CPUID fields for each logical processor within the same processor package. These values however can be different when comparing the values of logical processors on different packages.

- **Logical Processor Domain**—A properly configured platform can provide different values for these CPUID fields for each logical processor in the platform. The values contained within these may have their own scope as per a specific shared resource (i.e., cache, hybrid, etc.); in that case, each logical processor may need to be queried to obtain the full platform view of given features.

21.1.5 CPUID Runtime Mutable Fields

A CPUID field is said to be mutable if it can change during runtime. Such fields are affected by supervisor-mode operations that can affect processor mode, status bits, or privileged registers. Mutable fields that are expected to change dynamically as part of normal operation are shown in the table Table 21-1. Mutable fields that should remain consistent are shown in the table Table 21-2. Note all of the listed controls may not be available on all Intel processors.

Table 21-1. Runtime Mutable CPUID Fields Expected to Change During Normal Operation

Leaf	Sub-Leaf	Register	Field Name	Description and Mutability Control
01H	Ignored	ECX[27]	OSXSAVE	If 1, the OS has set CR4.OSXSAVE[bit 18] to enable XSETBV/XGETBV instructions to access XCRO and to support processor extended state management using XSAVE/XRSTOR.
07H	00H	ECX[4]	OSPKE	If 1, OS has set CR4.PKE to enable protection keys (and the RDPKRU/WRPKRU instructions).
0DH	00H	EBX[31:0]	XSAVE_BYTES_ENABLED_FEATURE	The size of the XSAVE/XRSTOR area required for the state bits enabled in XCRO.
0DH	01H	EBX[31:0]	XSAVE_BYTES_ENABLED_FEATURE	The size of the XSAVES/XRSTORS area required for the state bits enabled in XCRO and IA32_XSS.
19H	00H	EBX[0]	AESKLE	If 1, if the AES Key Locker instructions have been activated by system firmware and the OS has set CR4.KL[bit 19] = 1.
80000001H	Ignored	EDX[20]	SYSCALL_SYSRET_64	Intel processors support SYSCALL and SYSRET only in 64-bit mode. This feature flag is always enumerated as 0 outside 64-bit mode.

Table 21-2. Runtime Mutable CPUID Fields That Should Remain Consistent

Leaf	Sub-Leaf	Register	Field Name	Description and Mutability Control
00H	Ignored	EAX[31:0]	MAX_LEAF	Support for legacy software by limiting CPUID number of leaves reporting to a maximum of 2. This is set by using IA32_MISC_ENABLE[22] Limit CPUID Maxval.
01H	Ignored	ECX[3]	MONITOR	This feature flag reflects the setting in IA32_MISC_ENABLE[18] Enable Monitor FSM.
01H	Ignored	ECX[7]	EIST	This feature flag reflects the setting in IA32_MISC_ENABLE[16] Enhanced Intel SpeedStep Technology Enable.
01H	Ignored	EDX[9]	APIC	This feature flag reflects IA32_APIC_BASE[11], APIC Global Enable.
05H	Ignored	ECX[0]	MONITOR_MWAIT_EXTENSIONS	This field not available when CPUID.01H:ECX.MONITOR[3] = 0.

Table 21-2. Runtime Mutable CUID Fields That Should Remain Consistent (Contd.)

Leaf	Sub-Leaf	Register	Field Name	Description and Mutability Control
05H	Ignored	ECX[1]	INTERRUPT_AS_BREAK_EVENT	This field not available when CUID.01H:ECX.MONITOR[3] = 0.
05H	Ignored	EDX[15:0]	LARGEST_MONITOR_LINE_SIZE	This field not available when CUID.01H:ECX.MONITOR[3] = 0.
05H	Ignored	EAX[15:0]	SMALLEST_MONITOR_LINE_SIZE	This field not available when CUID.01H:ECX.MONITOR[3] = 0.

21.1.6 CUID Reserved Fields

Software must ignore and not rely upon the values returned by reserved fields of a CUID leaf or sub-leaf because they may have meaning on future processors. Once a previously-reserved field becomes defined, this specification will be updated to reflect that.

21.1.7 CUID Instruction for Serialization

Although the CUID instruction provides serialization, it is not the preferred method on newer processors that support the SERIALIZE instruction, which is enumerated via CUID.07H.00H:EDX[14]=1. If backward compatibility is required with older processors, use leaf 00H [CUID.00H] for serialization because it has the lowest latency when executed. See “Serializing Instructions” in Chapter 11 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A for more details.

21.1.8 IA32_BIOS_SIGN_ID Returns Microcode Update Signature

For processors that support the microcode update facility, the IA32_BIOS_SIGN_ID MSR is loaded with the update signature whenever CUID executes. The signature is returned in the upper DWORD. For details, see Chapter 11 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

21.2 METHODS FOR RETURNING BRANDING INFORMATION USING CUID

Use the following techniques to access branding information:

1. Processor brand string method.
2. Processor brand index; this method uses a software supplied brand string table.

These two methods are discussed in the following sections. For methods that are available in early processors, see Section 21.1.2, “Identification of Earlier Processors,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

21.2.1 The Processor Brand String Method

Figure 21-1 describes the algorithm used for detection of the brand string. Processor brand identification software should execute this algorithm on all Intel 64 and IA-32 processors.

This method (introduced with Pentium 4 processors) returns an ASCII brand identification string and the Processor Base frequency of the processor to the EAX, EBX, ECX, and EDX registers.

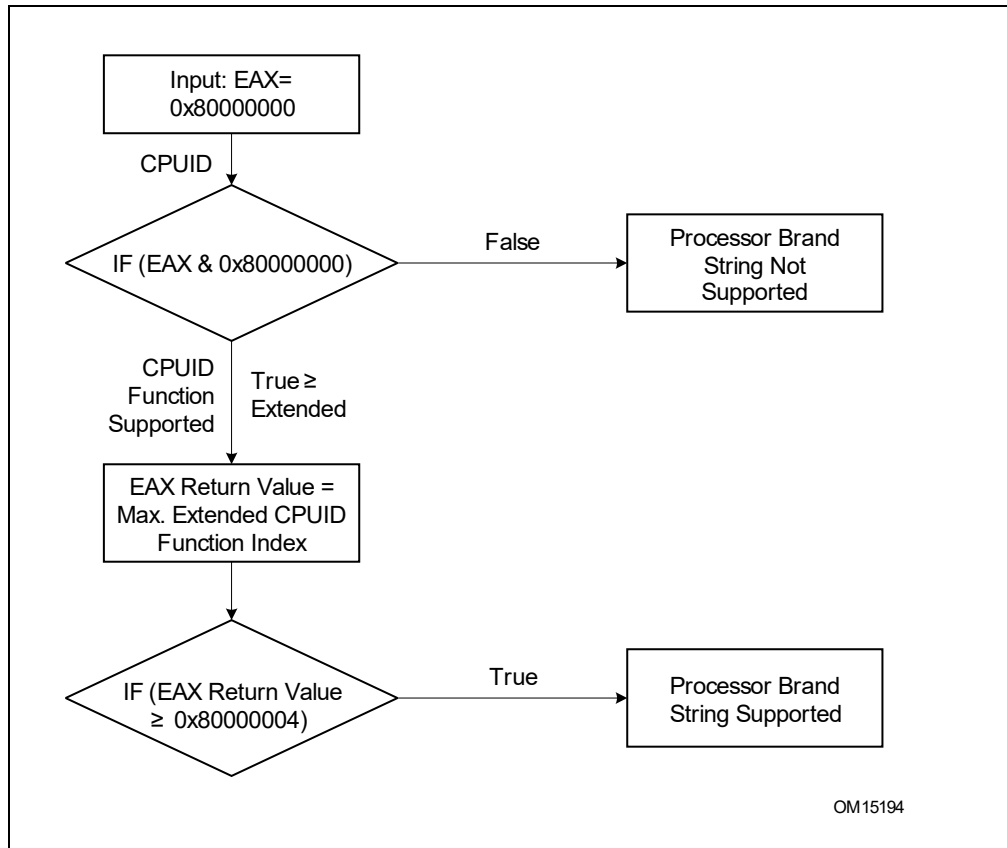


Figure 21-1. Determination of Support for the Processor Brand String

21.2.2 The Processor Brand Index Method

The brand index method (introduced with Pentium® III Xeon® processors) provides an entry point into a brand identification table that is maintained in memory by software. In this table, each brand index is associated with an ASCII brand identification string that identifies the official Intel family and model number of a processor.

When CPUID executes with EAX set to 1, the processor returns a brand index to the low byte in EBX. Software can then use this index to locate the brand identification string for the processor in the brand identification table. The first entry (brand index 0) in this table is reserved, allowing for backward compatibility with processors that do not support the brand identification feature. Starting with processor signature family ID = 0FH, model = 03H, brand index method is no longer supported. Use brand string method instead.

Table 21-3 shows brand indices that have identification strings associated with them.

Table 21-3. Mapping of Brand Indices; and Intel 64 and IA-32 Processor Brand Strings

Brand Index	Brand String
00H	This processor does not support the brand identification feature
01H	Intel® Celeron® processor ¹
02H	Intel® Pentium® III processor ¹
03H	Intel® Pentium® III Xeon® processor; If processor signature = 000006B1h, then Intel® Celeron® processor
04H	Intel® Pentium® III processor
06H	Mobile Intel® Pentium® III processor-M
07H	Mobile Intel® Celeron® processor ¹

Table 21-3. Mapping of Brand Indices; and Intel 64 and IA-32 Processor Brand Strings

08H	Intel® Pentium® 4 processor
09H	Intel® Pentium® 4 processor
0AH	Intel® Celeron® processor ¹
0BH	Intel® Xeon® processor; If processor signature = 00000F13h, then Intel® Xeon® processor MP
0CH	Intel® Xeon® processor MP
0EH	Mobile Intel® Pentium® 4 processor-M; If processor signature = 00000F13h, then Intel® Xeon® processor
0FH	Mobile Intel® Celeron® processor ¹
11H	Mobile Genuine Intel® processor
12H	Intel® Celeron® M processor
13H	Mobile Intel® Celeron® processor ¹
14H	Intel® Celeron® processor
15H	Mobile Genuine Intel® processor
16H	Intel® Pentium® M processor
17H	Mobile Intel® Celeron® processor ¹
18H – 0FFH	RESERVED

NOTES:

1. Indicates versions of these processors that were introduced after the Pentium III.

21.3 CPUID LEAVES

The remainder of this chapter provides CPUID enumeration information for Intel® 64 and IA-32 architectures.

CPUID.00H -- Maximum Input for Basic CPUID and Vendor ID

CPUID.00H returns the highest value the CPUID recognizes for returning basic processor information. The value is returned in the EAX register and is processor specific.

- This leaf is always valid.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-4. Leaf 00H Maximum Input for Basic CPUID and Vendor ID

Register	Field Name	Description	Domain
EAX[31:0]	MAX_LEAF	Maximum input value for basic CPUID Information.	Platform
EBX[31:0]	VENDOR_ID_1	"Genu"	Platform
ECX[31:0]	VENDOR_ID_2	"ntel"	Platform
EDX[31:0]	VENDOR_ID_3	"inel"	Platform

A vendor identification string is also returned in EBX, EDX, and ECX. For Intel processors, the string is "GenuineIntel" and is expressed:

EBX := 756e6547h (* "Genu", with G in the low eight bits of BL *)

EDX := 49656e69h (* "ineI", with i in the low eight bits of DL *)

ECX := 6c65746eh (* "ntel", with n in the low eight bits of CL *)

CPUID.01H -- Version and Features

CPUID.01H returns type, family, model, stepping, and feature information.

- This leaf is valid if MAX_LEAF $\geq$ 01H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-5. Leaf 01H Output Registers

CPUID Output Registers	Description
EAX[31:0]	Version information: Type, Family, Model, and Stepping ID (see “CPUID.01H:EAX—Version Information: Type, Family, Model and Stepping ID”).
EBX[31:0]	Feature information (see “CPUID.01H:EBX—Feature Information”).
ECX[31:0]	Feature information (see “CPUID.01H:ECX—Feature Information”).
EDX[31:0]	Feature information (see “CPUID.01H:EDX—Feature Information”).

CPUID.01H:EAX Version Information: Type, Family, Model and Stepping ID

The EAX register of CPUID.01H returns the information shown below.

Table 21-6. Leaf 01H Version and Features Returned in EAX

Register	Field Name	Description	Domain
EAX[3:0]	STEPPING_ID	Identifies a revision of the specific processor family and model. The stepping information is specified as a per-package basis for legacy processors. More recent processors do not allow mixing steppings.	Package
EAX[7:4]	MODEL_ID	Identifies a set of processors within a family. Certain models of Pentium® 4 processors allowed mixed Model IDs and would have this identified as a Package Domain.	Platform
EAX[11:8]	FAMILY_ID	Identifies a set of processors that have a general architectural similarity.	Platform
EAX[13:12]	PROCESSOR_TYPE	Identifies specific type of processor.	Platform
EAX[15:14]	Reserved	Reserved.	
EAX[19:16]	EXTENDED_MODEL_ID	When the Family ID is 06H or 0FH, this field is prepended to the Model ID to provide an 8-bit model identification.	Platform
EAX[27:20]	EXTENDED_FAMILY_ID	When the Family ID is 0FH, this field is added to the Family ID to provide an 8-bit family identification.	Platform
EAX[31:28]	Reserved	Reserved.	

CPUID.01H returns version information in EAX. For example: model, family, and processor type for the Intel Xeon processor 5100 series is as follows:

- Model — 1111B
- Family — 0101B
- Processor Type — 00B

See table below for available processor type values. Stepping IDs are provided as needed.

Table 21-7. Processor Type Field

Type	Encoding
Original OEM Processor	00B
Intel OverDrive® Processor	01B
Dual processor (not applicable to Intel486 processors)	10B
Intel reserved	11B

NOTE

See Section 21.1.2, “Identification of Earlier Processors,” for information on identifying earlier IA-32 processors. The Extended Family ID needs to be examined only when the Family ID is 0FH. Integrate the fields into a display using the following rule:

```

IF Family_ID ≠ 0FH THEN
    DisplayFamily = Family_ID;
ELSE
    DisplayFamily = Extended_Family_ID + Family_ID;
FI;

```

The Extended Model ID needs to be examined only when the Family ID is 06H or 0FH. Integrate the field into a display using the following rule:

```

IF (Family_ID = 06H or Family_ID = 0FH) THEN
    DisplayModel = (Extended_Model_ID « 4) + Model_ID;
ELSE
    DisplayModel = Model_ID;
FI;

```

CPUID.01H:EBX Feature information.

The EBX register of CPUID.01H returns the information shown below.

Table 21-8. Leaf 01H Version and Features Returned in EBX

Register	Field Name	Description	Domain
EBX[7:0]	BRAND_INDEX	This number provides an entry into a brand string table that contains brand strings for IA-32 processors. More information about this field is provided in Section 21.2.2, “The Processor Brand Index Method.”	Platform
EBX[15:8]	CLFLUSH_LINE_SIZE	Value * 8 = cache line size in bytes. This number indicates the size of the cache line flushed by the CLFLUSH and CLFLUSHOPT instructions in 8-byte increments. This field was introduced in the Pentium 4 processor.	Platform
EBX[23:16]	APIC_ID_SPACE	Maximum number of addressable IDs for logical processors in this physical package. The nearest power-of-2 integer that is not smaller than EBX[23:16] is the number of unique initial APIC IDs reserved for addressing different logical processors in a physical package. This field is only valid if CPUID.01H.EDX.HTT[28]= 1. See further details below on the usage of this field.	Platform

EBX[31:24]	INITIAL_APIC_ID	This number is the 8-bit ID that is assigned to the local APIC on the processor during power up. This field was introduced in the Pentium 4 processor. The 8-bit initial APIC ID in EBX[31:24] is replaced by the 32-bit x2APIC ID, available in Leaf 0BH and Leaf 1FH.	Logical Processor

The Maximum Addressable IDs for logical processors in this package should not be used on platforms that support CPUID leaf 0BH or CPUID leaf 1FH as it can be saturated and incorrect. Modern platforms can have many more processors than can be enumerated or have topology domains with discontinuous APIC ID reservations. To correctly enumerate APIC ID information on modern platforms, use CPUID.0BH or CPUID.1FH.

CPUID.01H:ECX Feature Information

The ECX register of CPUID.01H returns the information shown below. For all feature flags, a 1 indicates that the feature is supported. Software should identify Intel as the vendor to properly interpret feature flags. Software must confirm that a processor feature is present using feature flags returned by CPUID prior to using the feature. Software should not depend on future offerings retaining all features.

Table 21-9. Leaf 01H Version and Features Returned in ECX

Register	Field Name	Description	Domain
ECX[0]	SSE3	If 1, supports Streaming SIMD Extensions 3.	Platform
ECX[1]	PCLMULQDQ	If 1, supports the PCLMULQDQ instruction.	Platform
ECX[2]	DTES64	64-bit DS Area. If 1, supports DS area using 64-bit layout.	Platform
ECX[3]	MONITOR	If 1, supports the MONITOR/MWAIT and CPUID.05H.	Platform
ECX[4]	DS_CPL	If 1, supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.	Platform
ECX[5]	VMX	If 1, supports the Virtual Machine Extensions.	Platform
ECX[6]	SMX	If 1, supports Safer Mode Extensions. See Chapter 7, "Safer Mode Extensions Reference."	Platform
ECX[7]	EIST	If 1, supports Enhanced Intel SpeedStep® technology.	Platform
ECX[8]	TM2	If 1, supports Thermal Monitor 2.	Platform
ECX[9]	SSSE3	If 1, supports Supplemental Streaming SIMD Extensions 3.	Platform
ECX[10]	L1_CONTEXT_ID	If 1, the L1 data cache mode can be set to either adaptive mode or shared mode. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.	Platform
ECX[11]	DEBUG_INTERFACE	If 1, supports IA32_DEBUG_INTERFACE MSR for silicon debug.	Platform
ECX[12]	FMA	If 1, supports FMA extensions using YMM state.	Platform

ECX[13]	CMPXCHG16B	If 1, supports this instruction. See the “CMPXCHG8B/CMPXCHG16B—Compare and Exchange Bytes” section in this chapter for a description.	Platform
ECX[14]	XTPR_UPDATE_CONTROL	If 1, supports changing IA32_MISC_ENABLE[bit 23].	Platform
ECX[15]	PERF_CAPABILITIES	If 1, supports the performance and debug feature indication MSR IA32_PERF_CAPABILITIES.	Platform
ECX[16]	Reserved	Reserved.	
ECX[17]	PCID	If 1, supports Process-context identifiers and software setting CR4.PCIDE to 1.Process-context identifiers.	Platform
ECX[18]	DCA	If 1, supports the ability to prefetch data from a memory mapped device. See CPUID.09H.	Platform
ECX[19]	SSE4_1	If 1, supports SSE4.1.	Platform
ECX[20]	SSE4_2	If 1, supports SSE4.2.	Platform
ECX[21]	X2APIC	If 1, supports x2APIC feature.	Platform
ECX[22]	MOVBE	If 1, supports MOVBE instruction.	Platform
ECX[23]	POPCNT	If 1, supports the POPCNT instruction.	Platform
ECX[24]	TSC_DEADLINE	If 1, the processor’s local APIC timer supports one-shot operation using a TSC deadline value.	Platform
ECX[25]	AESNI	If 1, supports the AESNI instruction extensions.	Platform
ECX[26]	XSAVE	If 1, supports the XSAVE/XRSTOR processor extended states feature, the XSETBV/XGETBV instructions, and XCRO.	Platform
ECX[27]	OSXSAVE	If 1, the OS has set CR4.OSXSAVE[bit 18] to enable XSETBV/XGETBV instructions to access XCRO and to support processor extended state management using XSAVE/XRSTOR.	Logical Processor
ECX[28]	AVX	If 1, supports the AVX instruction extensions.	Platform
ECX[29]	F16C	If 1, supports 16-bit floating-point conversion instructions.	Platform
ECX[30]	RDRAND	If 1, supports RDRAND instruction.	Platform
ECX[31]	Not Used	Intel processors always return 0. Allocated for use by software emulation.	Platform

CPUID.01H:EDX Feature Information

The EDX register of CPUID.01H returns the information shown below. For all feature flags, a 1 indicates that the feature is supported. Software should identify Intel as the vendor to properly interpret feature flags. Software must confirm that a processor feature is present using feature flags returned by CPUID prior to using the feature. Software should not depend on future offerings retaining all features.

Table 21-10. Leaf 01H Version and Features Returned in EDX

Register	Field Name	Description	Domain
EDX[0]	FPU	Floating Point Unit On-Chip. The processor contains an x87 FPU.	Platform

EDX[1]	VME	If 1, supports Virtual 8086 mode enhancements, including CR4.VME for controlling the feature; CR4.PVI for protected mode virtual interrupts; software interrupt indirection; expansion of the TSS with the software indirection bitmap; and EFLAGS.VIF and EFLAGS.VIP flags.	Platform
EDX[2]	DE	If 1, supports I/O breakpoints debugging extensions, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.	Platform
EDX[3]	PSE	P If 1, supports page size extensions for large pages of size 4 MByte, including: CR4.PSE for controlling the feature; the defined dirty bit in PDE (Page Directory Entries); optional reserved bit trapping in CR3; PDEs; and PTEs.	Platform
EDX[4]	TSC	If 1, supports the Time Stamp Counter, RDTSC instruction, including CR4.TSD for controlling privilege.	Platform
EDX[5]	MSR	If 1, supports the Model Specific Registers RDMSR and WRMSR Instructions. Some of the MSRs are implementation dependent.	Platform
EDX[6]	PAE	If 1, supports the Physical Address Extension which is for physical addresses greater than 32 bits, including: extended page table entry formats; an extra level in the page translation tables; and 2-MByte pages rather than 4 Mbyte pages.	Platform
EDX[7]	MCE	If 1, supports exception 18 for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the modelspecific implementations of machine-check error logging, reporting, and processor shutdowns. Machine Check exception handlers may have to depend on processor version to do model specific processing of the exception, or test for the presence of the Machine Check feature.	Platform
EDX[8]	CMPXCHG8B	If 1, supports the CMPXCHG8B (64 bits) Instruction, implicitly locked and atomic.	Platform
EDX[9]	APIC	If 1, the processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FEE0000H to FEE00FFFH (by default - some processors permit the APIC to be relocated).	Platform
EDX[10]	Reserved	Reserved.	
EDX[11]	SEP	If 1 supports the SYSENTER and SYSEXIT Instructions and associated MSRs.	Platform

EDX[12]	MTRR	If 1, supports the Memory Type Range Registers. (The MTRRcap MSR contains feature bits that describe what memory types are supported, how many variable MTRRs are supported, and whether fixed MTRRs are supported.)	Platform
EDX[13]	PGE	If 1, supports the global bit in paging-structure entries that map a page, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.	Platform
EDX[14]	MCA	If 1, supports the Machine Check Architecture feature. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.	Platform
EDX[15]	CMOV	If 1, supports the Conditional Move Instructions. If CPUID.01H:EDX.FPU[0] (x87 FPU present) is 1 also, supports the FCOMI and FCMOV instructions.	Platform
EDX[16]	PAT	If 1, supports the Page Attribute Table feature. (This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory accessed through a linear address on a 4KB granularity.)	Platform
EDX[17]	PSE_36	If 1, supports the 36-Bit Page Size Extension which enables 4-MByte pages addressing physical memory beyond 4 GBytes with 32-bit paging. This feature indicates that upper bits of the physical address of a 4-MByte page are encoded in bits 20:13 of the page directory entry. Such physical addresses are limited by MAXPHYADDR and may be up to 40 bits in size.	Platform
EDX[18]	PSN	If 1, supports the 96-bit Processor Serial Number identification number feature, and the feature is enabled. Available only in Pentium III, see CPUID.03H.	Platform
EDX[19]	CLFLUSH	If 1, supports the CLFLUSH instruction.	Platform
EDX[20]	Reserved	Reserved.	
EDX[21]	DS	If 1, supports the Debug Store feature which provides the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and processor event-based sampling (PEBS) facilities (see Chapter 19, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B).	Platform

EDX[22]	ACPI	If 1, supports the Thermal Monitor and Software Controlled Clock Facilities. These are internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.	Platform
EDX[23]	MMX	If 1, supports the Intel MMX Technology.	Platform
EDX[24]	FXSR	If 1, supports the FXSAVE and FXRSTOR Instructions, which are fast save and restore of the floating-point context, and the availability of CR4.OSFXSR for an operating system to indicate support of same.	Platform
EDX[25]	SSE	If 1, supports SSE.	Platform
EDX[26]	SSE2	If 1, supports SSE2.	Platform
EDX[27]	SELF_SNOOP	If 1, supports Self Snoop which is the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.	Platform
EDX[28]	HTT	If 1, the value in CPUID.1.EBX[23:16] (the Maximum number of addressable IDs for logical processors in this package) is valid for the package. If 0, there is only a single logical processor in the package and software should assume only a single APIC ID is reserved.	Platform
EDX[29]	TM	If 1, supports the Thermal Monitor feature in which the processor implements the thermal monitor automatic thermal control circuitry (TCC). Thermal Monitor.	Platform
EDX[30]	Reserved	Reserved.	
EDX[31]	PBE	If 1, supports the Pending Break Enable feature, which is the use of the FERR#/PBE# pin when the processor is in the stop-clock state (STPCLK# is asserted) to signal the processor that an interrupt is pending and that the processor should return to normal operation to handle the interrupt.	Platform

CPUID.02H -- TLB/Cache/Prefetch Information

CPUID.02H returns TLB, cache, and prefetch information. This leaf has been superseded by CPUID.04H for cache enumeration and CPUID.18H for TLB enumeration. These processors will also report new descriptor values of types 0FEh or 0FFh to refer enumerations to CPUID.04H and CPUID.18H.

- This leaf is valid if MAX_LEAF $\geq$ 02H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-11. Leaf 02H TLB/Cache/Prefetch Information

Register	Field Name	Description	Domain
EAX[7:0]	Reserved	Reserved with a value of 1	
EAX[15:8]	DESCRIPTOR_1	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EAX[23:16]	DESCRIPTOR_2	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EAX[31:24]	DESCRIPTOR_3	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EBX[7:0]	DESCRIPTOR_4	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EBX[15:8]	DESCRIPTOR_5	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EBX[23:16]	DESCRIPTOR_6	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EBX[31:24]	DESCRIPTOR_7	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
ECX[7:0]	DESCRIPTOR_8	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
ECX[15:8]	DESCRIPTOR_9	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
ECX[23:16]	DESCRIPTOR_10	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
ECX[31:24]	DESCRIPTOR_11	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EDX[7:0]	DESCRIPTOR_12	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EDX[15:8]	DESCRIPTOR_13	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EDX[23:16]	DESCRIPTOR_14	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor
EDX[31:24]	DESCRIPTOR_15	See Table "Encoding of CPUID Leaf 2 Descriptors" below this table.	Logical Processor

CPUID.02H returns information about the processor's internal TLBs, cache, and prefetch hardware in the EAX, EBX, ECX, and EDX registers. The information is reported in encoded form and fall into the following categories:

- The least-significant byte in register EAX (register AL) will always return 01H. Software should ignore this value and not interpret it as an informational descriptor.
- The most significant bit (bit 31) of each register indicates whether the register contains valid information (set to 0) or is reserved (set to 1).
- If a register contains valid information, the information is contained in 1-byte descriptors. There are four types of encoding values for the byte descriptor, the encoding of these descriptors and encoding type are listed in the table

below.

Note that the order of descriptors in the EAX, EBX, ECX, and EDX registers is not defined; that is, specific bytes are not designated to contain descriptors for specific cache, prefetch, or TLB types. The descriptors may appear in any order. Note also a processor may report a general descriptor type FFH and not report any byte descriptor of “cache type” or “*TLB type” via CPUID.02H.

Table 21-12. Encoding of CPUID Leaf 2 Descriptors

Descriptor Value	Type	Cache or TLB Description
00H	General	Null descriptor, this byte contains no information.
01H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 32 entries.
02H	TLB	Instruction TLB: 4 MByte pages, fully associative, 2 entries.
03H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 64 entries.
04H	TLB	Data TLB: 4 MByte pages, 4-way set associative, 8 entries.
05H	TLB	Data TLB1: 4 MByte pages, 4-way set associative, 32 entries.
06H	Cache	1st-level instruction cache: 8 KBytes, 4-way set associative, 32 byte line size.
08H	Cache	1st-level instruction cache: 16 KBytes, 4-way set associative, 32 byte line size.
09H	Cache	1st-level instruction cache: 32KBytes, 4-way set associative, 64 byte line size.
0AH	Cache	1st-level data cache: 8 KBytes, 2-way set associative, 32 byte line size.
0BH	TLB	Instruction TLB: 4 MByte pages, 4-way set associative, 4 entries.
0CH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 32 byte line size.
0DH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 64 byte line size.
0EH	Cache	1st-level data cache: 24 KBytes, 6-way set associative, 64 byte line size.
1DH	Cache	2nd-level cache: 128 KBytes, 2-way set associative, 64 byte line size.
21H	Cache	2nd-level cache: 256 KBytes, 8-way set associative, 64 byte line size.
22H	Cache	3rd-level cache: 512 KBytes, 4-way set associative, 64 byte line size, 2 lines per sector.
23H	Cache	3rd-level cache: 1 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector.
24H	Cache	2nd-level cache: 1 MBytes, 16-way set associative, 64 byte line size.
25H	Cache	3rd-level cache: 2 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector.
29H	Cache	3rd-level cache: 4 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector.
2CH	Cache	1st-level data cache: 32 KBytes, 8-way set associative, 64 byte line size.
30H	Cache	1st-level instruction cache: 32 KBytes, 8-way set associative, 64 byte line size.
40H	Cache	No 2nd-level cache or, if processor contains a valid 2nd-level cache, no 3rd-level cache.
41H	Cache	2nd-level cache: 128 KBytes, 4-way set associative, 32 byte line size.
42H	Cache	2nd-level cache: 256 KBytes, 4-way set associative, 32 byte line size.
43H	Cache	2nd-level cache: 512 KBytes, 4-way set associative, 32 byte line size.
44H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 32 byte line size.
45H	Cache	2nd-level cache: 2 MByte, 4-way set associative, 32 byte line size.
46H	Cache	3rd-level cache: 4 MByte, 4-way set associative, 64 byte line size.
47H	Cache	3rd-level cache: 8 MByte, 8-way set associative, 64 byte line size.
48H	Cache	2nd-level cache: 3MByte, 12-way set associative, 64 byte line size.
49H	Cache	3rd-level cache: 4MB, 16-way set associative, 64-byte line size (Intel Xeon processor MP, Family 0FH, Model 06H) 2nd-level cache: 4 MByte, 16-way set associative, 64 byte line size.

4AH	Cache	3rd-level cache: 6MByte, 12-way set associative, 64 byte line size.
4BH	Cache	3rd-level cache: 8MByte, 16-way set associative, 64 byte line size.
4CH	Cache	3rd-level cache: 12MByte, 12-way set associative, 64 byte line size.
4DH	Cache	3rd-level cache: 16MByte, 16-way set associative, 64 byte line size.
4EH	Cache	2nd-level cache: 6MByte, 24-way set associative, 64 byte line size.
4FH	TLB	Instruction TLB: 4 KByte pages, 32 entries.
50H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 64 entries.
51H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 128 entries.
52H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 256 entries.
55H	TLB	Instruction TLB: 2-MByte or 4-MByte pages, fully associative, 7 entries.
56H	TLB	Data TLB0: 4 MByte pages, 4-way set associative, 16 entries.
57H	TLB	Data TLB0: 4 KByte pages, 4-way associative, 16 entries.
59H	TLB	Data TLB0: 4 KByte pages, fully associative, 16 entries.
5AH	TLB	Data TLB0: 2 MByte or 4 MByte pages, 4-way set associative, 32 entries.
5BH	TLB	Data TLB: 4 KByte and 4 MByte pages, 64 entries.
5CH	TLB	Data TLB: 4 KByte and 4 MByte pages, 128 entries.
5DH	TLB	Data TLB: 4 KByte and 4 MByte pages, 256 entries.
60H	Cache	1st-level data cache: 16 KByte, 8-way set associative, 64 byte line size.
61H	TLB	Instruction TLB: 4 KByte pages, fully associative, 48 entries.
63H	TLB	Data TLB: 2 MByte or 4 MByte pages, 4-way set associative, 32 entries and a separate array with 1 GByte pages, 4-way set associative, 4 entries.
64H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 512 entries.
66H	Cache	1st-level data cache: 8 KByte, 4-way set associative, 64 byte line size.
67H	Cache	1st-level data cache: 16 KByte, 4-way set associative, 64 byte line size.
68H	Cache	1st-level data cache: 32 KByte, 4-way set associative, 64 byte line size.
6AH	Cache	uTLB: 4 KByte pages, 8-way set associative, 64 entries.
6BH	Cache	DTLB: 4 KByte pages, 8-way set associative, 256 entries.
6CH	Cache	DTLB: 2M/4M pages, 8-way set associative, 128 entries.
6DH	Cache	DTLB: 1 GByte pages, fully associative, 16 entries.
70H	Cache	Trace cache: 12 K-?op, 8-way set associative.
71H	Cache	Trace cache: 16 K-?op, 8-way set associative.
72H	Cache	Trace cache: 32 K-?op, 8-way set associative.
76H	TLB	Instruction TLB: 2M/4M pages, fully associative, 8 entries.
78H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 64 byte line size.
79H	Cache	2nd-level cache: 128 KByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7AH	Cache	2nd-level cache: 256 KByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7BH	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7CH	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7DH	Cache	2nd-level cache: 2 MByte, 8-way set associative, 64 byte line size.
7FH	Cache	2nd-level cache: 512 KByte, 2-way set associative, 64-byte line size.

80H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64-byte line size.
82H	Cache	2nd-level cache: 256 KByte, 8-way set associative, 32 byte line size.
83H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 32 byte line size.
84H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 32 byte line size.
85H	Cache	2nd-level cache: 2 MByte, 8-way set associative, 32 byte line size.
86H	Cache	2nd-level cache: 512 KByte, 4-way set associative, 64 byte line size.
87H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size.
A0H	DTLB	DTLB: 4k pages, fully associative, 32 entries.
B0H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 128 entries.
B1H	TLB	Instruction TLB: 2M pages, 4-way, 8 entries or 4M pages, 4-way, 4 entries.
B2H	TLB	Instruction TLB: 4KByte pages, 4-way set associative, 64 entries.
B3H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 128 entries.
B4H	TLB	Data TLB1: 4 KByte pages, 4-way associative, 256 entries.
B5H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 64 entries.
B6H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 128 entries.
BAH	TLB	Data TLB1: 4 KByte pages, 4-way associative, 64 entries.
C0H	TLB	Data TLB: 4 KByte and 4 MByte pages, 4-way associative, 8 entries.
C1H	STLB	Shared 2nd-Level TLB: 4 KByte/2MByte pages, 8-way associative, 1024 entries.
C2H	DTLB	DTLB: 4 KByte/2 MByte pages, 4-way associative, 16 entries.
C3H	STLB	Shared 2nd-Level TLB: 4 KByte /2 MByte pages, 6-way associative, 1536 entries. Also 1GByte pages, 4-way, 16 entries.
C4H	DTLB	DTLB: 2M/4M Byte pages, 4-way associative, 32 entries.
CAH	STLB	Shared 2nd-Level TLB: 4 KByte pages, 4-way associative, 512 entries.
D0H	Cache	3rd-level cache: 512 KByte, 4-way set associative, 64 byte line size.
D1H	Cache	3rd-level cache: 1 MByte, 4-way set associative, 64 byte line size.
D2H	Cache	3rd-level cache: 2 MByte, 4-way set associative, 64 byte line size.
D6H	Cache	3rd-level cache: 1 MByte, 8-way set associative, 64 byte line size.
D7H	Cache	3rd-level cache: 2 MByte, 8-way set associative, 64 byte line size.
D8H	Cache	3rd-level cache: 4 MByte, 8-way set associative, 64 byte line size.
DCH	Cache	3rd-level cache: 1.5 MByte, 12-way set associative, 64 byte line size.
DDH	Cache	3rd-level cache: 3 MByte, 12-way set associative, 64 byte line size.
DEH	Cache	3rd-level cache: 6 MByte, 12-way set associative, 64 byte line size.
E2H	Cache	3rd-level cache: 2 MByte, 16-way set associative, 64 byte line size.
E3H	Cache	3rd-level cache: 4 MByte, 16-way set associative, 64 byte line size.
E4H	Cache	3rd-level cache: 8 MByte, 16-way set associative, 64 byte line size.
EAH	Cache	3rd-level cache: 12MByte, 24-way set associative, 64 byte line size.
EBH	Cache	3rd-level cache: 18MByte, 24-way set associative, 64 byte line size.
ECH	Cache	3rd-level cache: 24MByte, 24-way set associative, 64 byte line size.
F0H	Prefetch	64-Byte prefetching.
F1H	Prefetch	128-Byte prefetching.

FEH	General	CPUID leaf 2 does not report TLB descriptor information; use CPUID leaf 18H to query TLB and other address translation parameters.
FFH	General	CPUID leaf 2 does not report cache descriptor information, use CPUID leaf 4 to query cache parameters.

Example 21-1. Example of Cache and TLB Interpretation

The first member of the family of Pentium 4 processors returns the following information about caches and TLBs when the CPUID executes with an input value of 2:

EAX 66 5B 50 01H

EBX 0H

ECX 0H

EDX 00 7A 70 00H

Which means:

- The least-significant byte (byte 0) of register EAX is set to 01H. This value should be ignored.
- The most-significant bit of all four registers (EAX, EBX, ECX, and EDX) is set to 0, indicating that each register contains valid 1-byte descriptors.
- Bytes 1, 2, and 3 of register EAX indicate that the processor has:
 - 50H - a 64-entry instruction TLB, for mapping 4-KByte and 2-MByte or 4-MByte pages.
 - 5BH - a 64-entry data TLB, for mapping 4-KByte and 4-MByte pages.
 - 66H - an 8-KByte 1st level data cache, 4-way set associative, with a 64-Byte cache line size.
- The descriptors in registers EBX and ECX are valid, but contain NULL descriptors.
- Bytes 0, 1, 2, and 3 of register EDX indicate that the processor has:
 - 00H - NULL descriptor.
 - 70H - Trace cache: 12 K- μ op, 8-way set associative.
 - 7AH - a 256-KByte 2nd level cache, 8-way set associative, with a sectored, 64-byte cache line size.
 - 00H - NULL descriptor.

CPUID.03H -- Processor Serial Number

CPUID.03H returns the processor serial number, if available. Processor serial number (PSN) is not supported in the Pentium 4 processor or later.

- This leaf is valid if MAX_LEAF ≥ 03H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-13. Leaf 03H Processor Serial Number

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	PSN_31_0	Bits 00-31 of 96-bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)	Package
EDX[31:0]	PSN_63_32	Bits 32-63 of 96-bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)	Package

CPUID.04H -- Deterministic Cache Parameters

CPUID.04H returns the deterministic cache parameters for each cache level.

- This leaf is valid if CPUID.04H.00H:EAX[4:0] $\neq$ 0 and MAX_LEAF $\geq$ 04H.
- The sub-leaves are enumerated until sub-leaf n returns 0 in EAX[4:0].
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EAX[4:0] as 0.

Table 21-14. Leaf 04H Deterministic Cache Parameters

Register	Field Name	Description	Domain
EAX[4:0]	CACHE_TYPE	0 = Null, no more caches. 1 = Data Cache. 2 = Instruction Cache. 3 = Unified Cache. 4-31 = Reserved.	Logical Processor
EAX[7:5]	CACHE_LEVEL	Cache level (starts at 1).	Logical Processor
EAX[8]	SELF_INITIALIZING_CACHE	Self initializing cache level (does not need software initialization).	Logical Processor
EAX[9]	FULLY_ASSOC	Fully associative cache.	Logical Processor
EAX[13:10]	Reserved	Reserved.	
EAX[25:14]	MAX_LP_ADDRESSABLE_IDS	Maximum number of addressable IDs for logical processors sharing this cache. Add one to the return value to get the result. The nearest power-of-2 integer that is not smaller than (1 + EAX[25:14]) is the number of unique initial APIC IDs reserved for addressing different logical processors sharing this cache.	Logical Processor
EAX[31:26]	MAX_CORES_ADDRESSABLE_IDS_PKG	Maximum number of addressable IDs for processor cores in the physical package. Add one to the return value to get the result. The nearest power-of-2 integer that is not smaller than (1 + EAX[31:26]) is the number of unique Core_IDs reserved for addressing different processor cores in a physical package. Core ID is a subset of bits of the initial APIC ID. The returned value is constant for valid initial values in ECX. Valid ECX values start from 0. The maximum number of addressable IDs for processor cores in the physical package field may contain a saturated value and will not correctly identify the addressable ID reservations for cores in this package on processors where CPUID.0BH and/or CPUID.1FH exist. Processors which enumerate topology information in either CPUID.0BH or CPUID.1FH need to use those leaves to obtain the correct topology details.	Platform
EBX[11:0]	LINE_SIZE	System Coherency Line Size. Add one to the return value to get the result.	Platform
EBX[21:12]	PHYS_LINE_PARTITIONS	Physical line partitions. Add one to the return value to get the result.	Logical Processor
EBX[31:22]	NUM_WAYS	Ways of associativity. Add one to the return value to get the result.	Logical Processor

ECX[31:0]	NUM_SETS	Number of sets. Add one to the return value to get the result.	Logical Processor
EDX[0]	NOT_LWR_CACHE_FLUSH	0 = WBINVD/INVD from threads sharing this cache acts upon lower level caches for threads sharing this cache. 1 = WBINVD/INVD is not guaranteed to act upon lower level caches of non-originating threads sharing this cache.	Logical Processor
EDX[1]	INCLUSIVE_CACHE	0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels.	Logical Processor
EDX[2]	COMPLEX_CACHE_INDEXING	0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits.	Logical Processor
EDX[31:3]	Reserved	Reserved.	

When CPUID executes with EAX set to 04H and ECX contains an index value, the processor returns encoded data that describe a set of deterministic cache parameters (for the cache level associated with the input in ECX). Valid index values start from 0. Software can enumerate the deterministic cache parameters for each level of the cache hierarchy starting with an index value of 0, until the parameters report the value associated with the cache type field is 0.

This Cache Size in Bytes

$$= (\text{Ways} + 1) * (\text{Partitions} + 1) * (\text{Line\_Size} + 1) * (\text{Sets} + 1)$$

$$= (\text{EBX}[31:22] + 1) * (\text{EBX}[21:12] + 1) * (\text{EBX}[11:0] + 1) * (\text{ECX} + 1)$$

The CPUID.04H also reports data that can be used to derive the topology of processor cores in a physical package on legacy processors. This information is constant for all valid index values. Software can query the raw data reported by executing CPUID with EAX=04H and ECX=0 and use it as part of the topology enumeration algorithm on processors that do not enumerate either CPUID.0BH or CPUID.1FH as described in Chapter 10, “Multiple-Processor Management,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

CPUID.05H -- MONITOR and MWAIT Features

CPUID.05H returns the MONITOR and MWAIT feature information.

- This leaf is valid if MAX_LEAF ≥ 05H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-15. Leaf 05H MONITOR and MWAIT Features

Register	Field Name	Description	Domain
EAX[15:0]	SMALLEST_MONITOR_LINE_SIZE	Smallest monitor-line size in bytes (default is processor's monitor granularity).	Platform
EAX[31:16]	Reserved	Reserved.	
EBX[15:0]	LARGEST_MONITOR_LINE_SIZE	Largest monitor-line size in bytes (default is processor's monitor granularity).	Platform
EBX[31:16]	Reserved	Reserved.	
ECX[0]	MONITOR_MWAIT_EXTENSIONS	If 1, supports enumeration of MONITOR/MWAIT extensions (beyond EAX and EBX registers).	Platform
ECX[1]	INTERRUPT_AS_BREAK_EVENT	If 1, supports treating interrupts as break-event for MWAIT, even when interrupts are disabled.	Platform
ECX[31:2]	Reserved	Reserved.	
EDX[3:0]	C0_SUB_STATES	Number of C0* sub C-states supported using MWAIT.	Platform
EDX[7:4]	C1_SUB_STATES	Number of C1* sub C-states supported using MWAIT.	Platform
EDX[11:8]	C2_SUB_STATES	Number of C2* sub C-states supported using MWAIT.	Platform
EDX[15:12]	C3_SUB_STATES	Number of C3* sub C-states supported using MWAIT.	Platform
EDX[19:16]	C4_SUB_STATES	Number of C4* sub C-states supported using MWAIT.	Platform
EDX[23:20]	C5_SUB_STATES	Number of C5* sub C-states supported using MWAIT.	Platform
EDX[27:24]	C6_SUB_STATES	Number of C6* sub C-states supported using MWAIT.	Platform
EDX[31:28]	C7_SUB_STATES	Number of C7* sub C-states supported using MWAIT.	Platform

NOTE

The definition of C0 through C7 states for MWAIT extensions are processor-specific C-states, not ACPI C-state.

CPUID.05H returns information about features available to MONITOR/MWAIT instructions. The MONITOR instruction is used for address-range monitoring in conjunction with MWAIT instruction. The MWAIT instruction optionally provides additional extensions for advanced power management.

CPUID.06H -- Thermal and Power Management Features

CPUID.06H returns information about thermal and power management features.

- This leaf is valid if MAX_LEAF $\geq$ 06H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-16. Leaf 06H Thermal and Power Management Features

Register	Field Name	Description	Domain
EAX[0]	DIGITAL_TEMP_SENSOR	If 1, supports Digital Temperature Sensor.	Platform
EAX[1]	TURBO_BOOST	If 1, supports Intel Turbo Boost Technology. (see description of IA32_MISC_ENABLE[38]).	Platform
EAX[2]	ALWAYS_RUNNING_APIC_TIMER	If 1, supports APIC-Timer-always-running feature.	Platform
EAX[3]	Reserved	Reserved.	
EAX[4]	POWER_LIMIT_NOTIFY	If 1, supports power limit notification controls.	Platform
EAX[5]	EXT_CLOCK_MOD	If 1, supports clock modulation duty cycle extension.	Platform
EAX[6]	PKG_THERM_MGMT	If 1, supports package thermal management.	Platform
EAX[7]	HWP	If 1, supports HWP base registers (IA32_PM_ENABLE[bit 0], IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS).	Platform
EAX[8]	HWP_INTERRUPT	If 1, supports IA32_HWP_INTERRUPT MSR.	Platform
EAX[9]	HWP_ACTIVITY_WINDOW	If 1, supports IA32_HWP_REQUEST[bits 41:32].	Platform
EAX[10]	HWP_EPP	If 1, supports IA32_HWP_REQUEST[bits 31:24].	Platform
EAX[11]	HWP_REQUEST_PKG	If 1, supports IA32_HWP_REQUEST_PKG MSR.	Platform
EAX[12]	Reserved	Reserved.	
EAX[13]	HDC	If 1, supports HDC base registers IA32_PKG_HDC_CTL, IA32_PM_CTL1, and IA32_THREAD_STALL MSRs.	Platform
EAX[14]	TURBO_BOOST_MAX	If 1, supports Intel® Turbo Boost Max Technology 3.0.	Platform
EAX[15]	HWP_CAP	If 1, supports Highest Performance change capability.	Platform
EAX[16]	HWP_PECI_OVERRIDE	If 1, supports HWP Peci override.	Platform
EAX[17]	FLEXIBLE_HWP	If 1, supports Flexible HWP.	Platform
EAX[18]	HWP_REQUEST_FAST_ACCESS	If 1, supports Fast access mode for the IA32_HWP_REQUEST MSR.	Platform
EAX[19]	HW_FEEDBACK	If 1, supports IA32_HW_FEEDBACK_PTR MSR, IA32_HW_FEEDBACK_CONFIG MSR, IA32_PACKAGE_THERM_STATUS MSR bit 26, and IA32_PACKAGE_THERM_INTERRUPT MSR bit 25.	Platform
EAX[20]	HWP_REQUEST_IGNORE_IDLE	If 1, supports Ignoring Idle Logical Processor HWP request.	Platform
EAX[21]	Reserved	Reserved.	
EAX[22]	HWP_CTL	If 1, supports IA32_HWP_CTL MSR.	Platform

EAX[23]	THREAD_DIRECTOR	If 1, supports Intel® Thread Director. IA32_HW_FEEDBACK_CHAR and IA32_HW_FEEDBACK_THREAD_CONFIG MSRs are supported if set.	Platform
EAX[31:24]	Reserved	Reserved.	
EBX[3:0]	DTS_NUM_INT_THRESHOLDS	Number of Interrupt Thresholds in Digital Thermal Sensor.	Platform
EBX[31:4]	Reserved	Reserved.	
ECX[0]	HW_FEEDBACK_CAP	If 1, supports IA32_MPERF and IA32_APERF which provide a measure of delivered processor performance (since last reset of the counters), as a percentage of the expected processor performance when running at the TSC frequency.	Platform
ECX[2:1]	Reserved	Reserved.	
ECX[3]	ENERGY_PERF_BIAS	If 1, supports performance-energy bias preference and a new architectural MSR called IA32_ENERGY_PERF_BIAS (1BOH).	Platform
ECX[7:4]	Reserved	Reserved.	
ECX[15:8]	HW_FEEDBACK_NUM_CLASSES	Number of Intel® Thread Director classes supported by the processor. Information for that many classes is written into the Intel Thread Director Table by the hardware.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[7:0]	HW_FEEDBACK_CAPS	Bitmap of supported hardware feedback interface capabilities. 0 = If 1, supports performance capability reporting. 1 = If 1, supports energy efficiency capability reporting. 2-7 = Reserved. Bits 0 and 1 will always be set together.	Package
EDX[11:8]	HW_FEEDBACK_TABLE_SIZE	Enumerates the size of the hardware feedback interface structure in number of 4 KB pages. Add one to the return value to get the result.	Package
EDX[15:12]	Reserved	Reserved.	
EDX[31:16]	HW_FEEDBACK_TABLE_INDEX	Index (starting at 0) of this logical processor's row in the hardware feedback interface structure. Note that on some parts the index may be same for multiple logical processors. On some parts the indices may not be contiguous, i.e., there may be unused rows in the hardware feedback interface structure.	Logical Processor

Details around these features are described in Chapter 16, "Power and Thermal Management," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

CPUID.07H -- Structured Extended Feature Flags

CPUID.07H returns structured extended feature flags enumeration information. The sub-sections of Section provide leaf 07H information.

- This leaf is valid if MAX_LEAF $\geq$ 07H.
- The maximum sub-leaf value for ECX is specified in CPUID.07H.00H.EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid Sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.07H.00H -- Structured Extended Feature Flags Main Sub-Leaf

CPUID.07H.00H returns the maximum input value of the highest leaf 07H sub-leaf; and EBX, ECX, and EDX contain information of extended feature flags.

Table 21-17. Leaf 07H Sub-Leaf (ECX=0) Output Registers

CPUID Output Registers	Field Name	Description	CPUID Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum input value for supported leaf 07H subleaves.	Platform
EBX[31:0]		Extended Feature Flags Information in EBX (see "CPUID.07H.00H:EBX—Extended Feature Flags Information")	
ECX[31:0]		Extended Feature Flags Information in ECX (see "CPUID.07H.00H:ECX—Extended Feature Flags Information")	
EDX[31:0]		Extended Feature Flags Information in EDX (see "CPUID.07H.00H:EDX—Extended Feature Flags Information")	

Table 21-18. Leaf 07H.00H Structured Extended Feature Flags Returned in EAX

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum input value for supported leaf 07H sub-leaves.	Platform

CPUID.07H.00H:EBX Extended Feature Flags Information

The EBX register of CPUID.07H.00H returns the information shown below.

Table 21-19. Leaf 07H.00H Structured Extended Feature Flags Returned in EBX

Register	Field Name	Description	Domain
EBX[0]	FSGSBASE	If 1, supports RDFSBASE/RDGSBASE/WRFS-BASE/WRGSBASE.	Platform
EBX[1]	TSC_ADJUST	If 1, the IA32_TSC_ADJUST MSR is supported.	Platform
EBX[2]	SGX	If 1, supports Intel® Software Guard Extensions (Intel® SGX Extensions).	Platform
EBX[3]	BMI1	If 1, supports the BMI1 instructions.	Platform
EBX[4]	HLE	If 1, supports the Hardware Lock Elision instruction set.	Platform
EBX[5]	AVX2	If 1, supports Intel® Advanced Vector Extensions 2 (Intel® AVX2).	Platform

EBX[6]	FDP_EXCPTN_ONLY	If 1, the x87 FPU Data Pointer is updated only on x87 exceptions.	Platform
EBX[7]	SMEP	If 1, supports Supervisor-Mode Execution Prevention.	Platform
EBX[8]	BMI2	If 1, supports the BMI2 instructions.	Platform
EBX[9]	ENH_REP_MOVSB_STOSB	If 1, supports Enhanced REP MOVSB/STOSB.	Platform
EBX[10]	INVPCID	If 1, supports INVPCID instruction for system software that manages process-context identifiers.	Platform
EBX[11]	RTM	If 1, supports the Restricted Transactional Memory instruction set.	Platform
EBX[12]	RDT_M	If 1, supports Intel® Resource Director Technology (Intel® RDT) Monitoring capability.	Platform
EBX[13]	FCS_FDS_DEPRECATION	If 1, deprecates FPU CS and FPU DS values.	Platform
EBX[14]	MPX	If 1, supports Intel® Memory Protection Extensions.	Platform
EBX[15]	RDT_A	If 1, supports Intel® Resource Director Technology (Intel® RDT) Allocation capability.	Platform
EBX[16]	AVX512F	If 1, supports the AVX512F instructions.	Platform
EBX[17]	AVX512DQ	If 1, supports the AVX512DQ instructions.	Platform
EBX[18]	RDSEED	If 1, supports the RDSEED instruction.	Platform
EBX[19]	ADX	If 1, supports the ADX instructions.	Platform
EBX[20]	SMAP	If 1, supports Supervisor-Mode Access Prevention and the CLAC/STAC instructions.	Platform
EBX[21]	AVX512_IFMA	If 1, supports the AVX512_IFMA instructions.	Platform
EBX[22]	Reserved	Reserved.	
EBX[23]	CLFLUSHOPT	If 1, supports the CLFLUSHOPT instruction.	Platform
EBX[24]	CLWB	If 1, supports the CLWB instruction.	Platform
EBX[25]	INTEL_PROC_TRACE	If 1, supports Intel® Processor Trace.	Platform
EBX[26]	AVX512PF	If 1, supports the AVX512PF instructions. (Intel® Xeon Phi™ only.)	Platform
EBX[27]	AVX512ER	If 1, supports the AVX512ER instructions. (Intel® Xeon Phi™ only.)	Platform
EBX[28]	AVX512CD	If 1, supports the AVX512CD instructions.	Platform
EBX[29]	SHA	If 1, supports Intel® Secure Hash Algorithm Extensions (Intel® SHA Extensions).	Platform
EBX[30]	AVX512BW	If 1, supports the AVX512BW instructions.	Platform
EBX[31]	AVX512VL	If 1, supports the AVX512VL instructions.	Platform

CPUID.07H.00H:ECX Extended Feature Flags Information

The ECX register of CPUID.07H.00H returns the information shown below.

Table 21-20. Leaf 07H.00H Structured Extended Feature Flags Returned in ECX

Register	Field Name	Description	Domain

ECX[0]	PREFETCHWT1	If 1, supports the PREFETCHWT1 instruction. (Intel® Xeon Phi™ only.)	Platform
ECX[1]	AVX512_VBMI	If 1, supports the AVX512_VBMI instructions.	Platform
ECX[2]	UMIP	If 1, supports user-mode instruction prevention.	Platform
ECX[3]	PKU	If 1, supports protection keys for user-mode pages.	Platform
ECX[4]	OSPKE	If 1, the OS has set CR4.PKE to enable protection keys and the RDPKRU/WRPKRU instructions.	Logical Processor
ECX[5]	WAITPKG	If 1, supports the TPAUSE, UMONITOR, and UMWAIT instructions.	Platform
ECX[6]	AVX512_VBMI2	If 1, supports the AVX512_VBMI2 instructions.	Platform
ECX[7]	CET_SS	If 1, supports CET shadow stack features. Processors that set this bit define bits 1:0 of the IA32_U_CET and IA32_S_CET MSRs. Enumerates support for the following MSRs: IA32_INTERRUPT_SPP_TABLE_ADDR, IA32_PL3_SSP, IA32_PL2_SSP, IA32_PL1_SSP, and IA32_PLO_SSP.	Platform
ECX[8]	GFNI	If 1, supports the GFNI instruction set.	Platform
ECX[9]	VAES	If 1 and Intel AVX supported, supports the VEX-encoded AES instruction set.	Platform
ECX[10]	VPCLMULQDQ	If 1 and Intel AVX supported, supports the VPCLMULQDQ instruction.	Platform
ECX[11]	AVX512_VNNI	If 1, supports the AVX512_VNNI instructions.	Platform
ECX[12]	AVX512_BITALG	If 1, supports the AVX512_BITALG instructions.	Platform
ECX[13]	TME_EN	If 1, the following MSRs are supported: IA32_TME_CAPABILITY, IA32_TME_ACTIVATE, IA32_TME_EXCLUDE_MASK, and IA32_TME_EXCLUDE_BASE.	Platform
ECX[14]	AVX512_VPOPCNTDQ	If 1, supports the AVX512_VPOPCNTDQ instructions.	Platform
ECX[15]	Reserved	Reserved.	
ECX[16]	LA57	If 1, supports 57-bit linear addresses and five-level paging.	Platform
ECX[21:17]	MPX_MAWAU	The value of MAWAU used by the BNDLDX and BNDSTX instructions in 64-bit mode.	Platform
ECX[22]	RDPID	If 1, RDPID and the IA32_TSC_AUX MSR are available.	Platform
ECX[23]	KEY_LOCKER	If 1, supports Key Locker.	Platform
ECX[24]	BUS_LOCK_DETECT	If 1, indicates support for OS bus-lock detection.	Platform
ECX[25]	CLDEMOTE	If 1, supports cache line demote.	Platform
ECX[26]	Reserved	Reserved.	
ECX[27]	MOVDIRI	If 1, supports the MOVDIRI instruction.	Platform
ECX[28]	MOVDIR64B	If 1, supports the MOVDIR64B instruction.	Platform
ECX[29]	ENQCMD	If 1, supports Enqueue Stores.	Platform

ECX[30]	SGX_LC	If 1, supports SGX Launch Configuration.	Platform
ECX[31]	PKS	If 1, supports protection keys for supervisor-mode pages.	Platform

CPUID.07H.00H:EDX Extended Feature Flags Information

The EDX register of CPUID.07H.00H returns the information shown below.

Table 21-21. Leaf 07H.00H Structured Extended Feature Flags Returned in EDX

Register	Field Name	Description	Domain
EDX[0]	Reserved	Reserved.	
EDX[1]	SGX_KEYS	If 1, supports Attestation Services for Intel® SGX.	Platform
EDX[2]	AVX512_4VNNIW	If 1, supports the AVX512_4VNNIW instructions. (Intel® Xeon Phi™ only.)	Platform
EDX[3]	AVX512_4FMAPS	If 1, supports the AVX512_4FMAPS instructions. (Intel® Xeon Phi™ only.)	Platform
EDX[4]	FAST_SHORT_REP_MOVSB	If 1, supports Fast Short REP MOVSB.	Platform
EDX[5]	UINTR	If 1, supports user interrupts.	Platform
EDX[7:6]	Reserved	Reserved.	
EDX[8]	AVX512_VP2INTERSECT	If 1, supports the AVX512_VP2INTERSECT instruction.	Platform
EDX[9]	MCU_OPT_CTRL	If 1, supports both the IA32_MCU_OPT_CTRL MSR and its bit 0 (RNGDS_MITG_DIS).	Platform
EDX[10]	MD_CLEAR	If 1, supports MD_CLEAR.	Platform
EDX[11]	RTM_ALWAYS_ABORT	If 1, any execution of XBEGIN immediately aborts and transitions to the specified fallback address.	Platform
EDX[12]	Reserved	Reserved.	
EDX[13]	RTM_FORCE_ABORT	If 1, supports RTM_FORCE_ABORT and the IA32_TSX_FORCE_ABORT MSR. These allow software to set IA32_TSX_FORCE_ABORT[0] (RTM_FORCE_ABORT).	Platform
EDX[14]	SERIALIZE	If 1, supports SERIALIZE instruction.	Platform
EDX[15]	HYBRID	If 1, the processor is identified as a hybrid part. If CPUID.00H.MAXLEAF ≥ 1AH and CPUID.1AH:EAX <> 0, then the Native Model ID Enumeration Leaf 1AH exists.	Platform
EDX[16]	TSXLDTRK	If 1, supports Intel TSX suspend/resume of load address tracking.	Platform
EDX[17]	Reserved	Reserved.	
EDX[18]	PCONFIG	If 1, supports the PCONFIG instruction.	Platform
EDX[19]	ARCH_LBRS	If 1, supports architectural LBRS.	Platform
EDX[20]	CET_IBT	If 1, supports CET indirect branch tracking features. Processors that set this bit define bits 5:2 and bits 63:10 of the IA32_U_CET and IA32_S_CET MSRs.	Platform

EDX[21]	Reserved	Reserved.	
EDX[22]	AMX_BF16	If 1, supports tile computational operations on bfloat16 numbers.	Platform
EDX[23]	AVX512_FP16	If 1, supports the FP16 data type with AVX512 instructions.	Platform
EDX[24]	AMX_TILE	If 1, supports tile architecture.	Platform
EDX[25]	AMX_INT8	If 1, supports tile computational operations on 8-bit integers.	Platform
EDX[26]	IBRS_IBPB	If 1, supports indirect branch restricted speculation (IBRS) and the indirect branch predictor barrier (IBPB). Processors that set this bit support the IA32_SPEC_CTRL MSR and the IA32_PRED_CMD MSR. They allow software to set IA32_SPEC_CTRL[0] (IBRS) and IA32_PRED_CMD[0] (IBPB).	Platform
EDX[27]	SPEC_CTRL_ST_PREDICTORS	If 1, supports single thread indirect branch predictors (STIBP). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[1] (STIBP).	Platform
EDX[28]	L1D_FLUSH_INTERFACE	If 1, supports L1D_FLUSH. Processors that set this bit support the IA32_FLUSH_CMD MSR. They allow software to set IA32_FLUSH_CMD[0] (L1D_- FLUSH).	Platform
EDX[29]	ARCH_CAPABILITIES	If 1, supports the IA32_ARCH_CAPABILITIES MSR.	Platform
EDX[30]	CORE_CAPABILITIES	If 1, supports the IA32_CORE_CAPABILITIES MSR. IA32_CORE_CAPABILITIES is an architectural MSR that enumerates model-specific features. A bit being set in this MSR indicates that a model specific feature is supported; software must still consult CPUID family/model/stepping to determine the behavior of the enumerated feature as features enumerated in IA32_CORE_CAPABILITIES may have different behavior on different processor models. Some of these features may have behavior that is consistent across processor models (and for which consultation of CPUID family/model/stepping is not necessary); such features are identified explicitly where they are documented in this manual.	Platform
EDX[31]	SPEC_CTRL_SSBD	If 1, supports Speculative Store Bypass Disable (SSBD). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[2] (SSBD).	Platform

CPUID.07H.01H -- Structured Extended Feature Sub-Leaf 1

CPUID.07H.01H:EAX Extended Feature Information

The EAX register of CPUID.07H.01H returns the information shown below.

Table 21-22. Leaf 07H.01H Structured Extended Feature Flags Returned in EAX

Register	Field Name	Description	Domain
EAX[0]	SHA512	If 1, supports the SHA512 instructions.	Platform
EAX[1]	SM3	If 1, supports the SM3 instructions.	Platform
EAX[2]	SM4	If 1, supports the SM4 instructions.	Platform
EAX[3]	Reserved	Reserved.	
EAX[4]	AVX_VNNI	If 1, supports the VEX-encoded versions of the Vector Neural Network Instructions.	Platform
EAX[5]	AVX512_BF16	If 1, supports the Vector Neural Network Instructions supporting BFLOAT16 inputs and conversion instructions from IEEE single precision.	Platform
EAX[6]	LASS	If 1, supports Linear Address Space Separation.	Platform
EAX[7]	CMPCCXADD	If 1, supports the CMPccXADD instruction.	Platform
EAX[8]	ARCH_PERFMON_EXT	If 1, supports ArchPerfmonExt. When set, indicates that the Architectural Performance Monitoring Extended Leaf (EAX=23H) is valid.	Platform
EAX[9]	Reserved	Reserved.	
EAX[10]	FAST_REP_MOVSB	If 1, supports fast zero-length REP MOVSB.	Platform
EAX[11]	FAST_REP_STOSB	If 1, supports fast short REP STOSB.	Platform
EAX[12]	FAST_REP_CMPSB_SCASB	If 1, supports fast short REP CMPSB, REP SCASB.	Platform
EAX[16:13]	Reserved	Reserved.	
EAX[17]	FRED	If 1, supports Flexible Return and Event Delivery and the architectural state (MSRs) defined by FRED. Any Intel processor that enumerates support for FRED transitions will also enumerate support for LKGS.	Platform
EAX[18]	LKGS	If 1, supports the LKGS (load into IA32_KERNEL_GS_BASE) instruction.	Platform
EAX[19]	WRMSRNS	If 1, supports the WRMSRNS instruction.	Platform
EAX[20]	Reserved	Reserved.	
EAX[21]	AMX_FP16	If 1, supports tile computational operations on FP16 numbers.	Platform
EAX[22]	HRESET	If 1, supports history reset via the HRESET instruction and the IA32_HRESET_ENABLE MSR. When set, indicates that the Processor History Reset Leaf (EAX = 20H) is valid.	Platform
EAX[23]	AVX_IFMA	If 1, supports the AVX-IFMA instructions.	Platform
EAX[25:24]	Reserved	Reserved.	
EAX[26]	LAM	If 1, supports Linear Address Masking.	Platform
EAX[27]	MSRLIST	If 1, supports the RDMSRLIST and WRMSRLIST instructions and the IA32_BARRIER MSR.	Platform
EAX[29:28]	Reserved	Reserved.	
EAX[30]	INVD_DISABLE_POST_BIOS_DONE	If 1, supports INVD execution prevention after BIOS Done.	Platform

EAX[31]	Reserved	Reserved.	

CPUID.07H.01H:EBX Extended Feature Information

The EBX register of CPUID.07H.01H returns the information shown below.

Table 21-23. Leaf 07H.01H Structured Extended Feature Flags Returned in EBX

Register	Field Name	Description	Domain
EBX[0]	PPIN	If 1, supports the IA32_PPIN and IA32_PPIN_CTL MSRs.	Platform
EBX[1]	PBNDKB	If 1, supports the PBNDKB instruction and enumerates the existence of the IA32_TSE_CAPABILITY MSR.	Platform
EBX[2]	Reserved	Reserved.	
EBX[3]	CPUIDMAXVAL_LIM_RMV	If 1, IA32_MISC_ENABLE[bit 22] cannot be set to 1 to limit the value returned by CPUID.00H:EAX[7:0].	Platform
EBX[31:4]	Reserved	Reserved.	

CPUID.07H.01H:ECX Extended Feature Information

The ECX register of CPUID.07H.01H returns the information shown below.

Table 21-24. Leaf 07H.01H Structured Extended Feature Flags Returned in ECX

Register	Field Name	Description	Domain
ECX[31:0]	Reserved	Reserved.	

CPUID.07H.01H:EDX Extended Feature Information

The EDX register of CPUID.07H.01H returns the information shown below.

Table 21-25. Leaf 07H.01H Structured Extended Feature Flags Returned in EDX

Register	Field Name	Description	Domain
EDX[3:0]	Reserved	Reserved.	
EDX[4]	AVX_VNNI_INT8	If 1, supports the AVX-VNNI-INT8 instructions.	Platform
EDX[5]	AVX_NE_CONVERT	If 1, supports the AVX-NE-CONVERT instructions.	Platform
EDX[9:6]	Reserved	Reserved.	
EDX[10]	AVX_VNNI_INT16	If 1, supports the AVX-VNNI-INT16 instructions.	Platform
EDX[13:11]	Reserved	Reserved.	
EDX[14]	PREFETCHI	If 1, supports the PREFETCHIT0/1 instructions.	Platform
EDX[16:15]	Reserved	Reserved.	
EDX[17]	UIRET_UIF	If 1, UIRET sets UIF to the value of bit 1 of the RFLAGS image loaded from the stack.	Platform

EDX[18]	CET_SSS	If 1, indicates that an operating system can enable supervisor shadow stacks as long as it ensures that a supervisor shadow stack cannot become prematurely busy due to page faults (see Section 17.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1). When emulating the CPUID instruction, a virtual-machine monitor (VMM) should return this bit as 1 only if it ensures that VM exits cannot cause a guest supervisor shadow stack to appear to be prematurely busy. Such a VMM could set the "prematurely busy shadow stack" VM-exit control and use the additional information that it provides.	Platform
EDX[19]	AVX10	If 1, supports the Intel® AVX10 instructions and indicates the presence of CPUID.24H, which enumerates the version number.	Platform
EDX[21:20]	Reserved	Reserved.	
EDX[22]	SEC-TEE-ATTESTATION	N/A	Platform
EDX[23]	MWAIT	If 1, MWAIT is supported (even if CPUID.01H:ECX.MONITOR[3] is enumerated as 0).	Platform
EDX[24]	SLSM	Static LSM is supported on this platform. If set, IA32_INTEGRITY_STATUS (0x2DC) is available for software use.	Platform
EDX[31:25]	Reserved	Reserved.	

CPUID.07H.02H -- Structured Extended Feature Sub-Leaf 2

CPUID.07H.02H returns the structured extended feature information contained in the sub-sections of this section.

Table 21-26. Leaf 07H Sub-Leaf (ECX=2) Output Registers

CPUID Output Registers	Description
EAX[31:0]	Reserved
EBX[31:0]	Reserved
ECX[31:0]	Reserved
EDX[31:0]	Extended Feature Information (see "CPUID.07H.02H:EDX—Extended Feature Information")

CPUID.07H.02H:EDX Extended Feature Information

The EDX register of CPUID.07H.02H returns the information shown below.

Table 21-27. CPUID.07H.02H Extended Feature Information Provided in EDX¹

Register	Field Name	Description	Domain
EDX[0]	PSFD	If 1, supports bit 7 of the IA32_SPEC_CTRL MSR. Bit 7 of this MSR disables Fast Store Forwarding Predictor without disabling Speculative Store Bypass.	Platform

EDX[1]	IPRED_CTRL	If 1, supports bits 3 and 4 of the IA32_SPEC_CTRL MSR. Bit 3 of this MSR enables IPRED_DIS control for CPL3. Bit 4 of this MSR enables IPRED_DIS control for CPL0/1/2.	Platform
EDX[2]	RRSBA_CTRL	If 1, supports bits 5 and 6 of the IA32_SPEC_CTRL MSR. Bit 5 of this MSR disables RRSBA behavior for CPL3. Bit 6 of this MSR disables RRSBA behavior for CPL0/1/2.	Platform
EDX[3]	DDPD_U	If 1, supports bit 8 of the IA32_SPEC_CTRL MSR. Bit 8 of this MSR disables Data Dependent Prefetcher.	Platform
EDX[4]	BHI_CTRL	If 1, supports bit 10 of the IA32_SPEC_CTRL MSR. Bit 10 of this MSR enables BHI_DIS_S behavior.	Platform
EDX[5]	MCDT_NO	If 1, the processor does not exhibit MXCSR Configuration Dependent Timing (MCDT) behavior and does not need to be mitigated to avoid data- dependent behavior for certain instructions.	Platform
EDX[6]	UC_LOCK_DISABLE	If 1, supports the UC-lock disable feature and it causes #AC.	Platform
EDX[7]	MONITOR_MITG_NO	If 1, the MONITOR/UMONITOR instructions are not affected by performance or power issues due to MONITOR/UMONITOR instructions exceeding the capacity of an internal monitor tracking table. If 0, then the product may be affected by this issue.	Platform
EDX[31:8]	Reserved	Reserved.	

NOTE

1. Leaf 07H output depends on the initial value in ECX. If ECX contains an invalid sub-leaf index, EDX returns 0.

CPUID.08H -- Reserved

This leaf is reserved.

Table 21-28. Leaf 08H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.09H -- Direct Cache Access Information

- CPUID.09H returns information about Direct Cache Access capabilities.
- This leaf is valid if CPUID.01H:ECX.DCA[18] = 1 and MAX_LEAF ≥ 09H.
 - This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-29. Leaf 09H Direct Cache Access Information

Register	Field Name	Description	Domain
EAX[31:0]	PLATFORM_DCA_CAP	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H).	Platform
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

The IA32_PLATFORM_DCA_CAP MSR is valid when CPUID.01H:ECX.DCA[18] = 1.

CPUID.0AH -- Architectural Performance Monitoring

CPUID.0AH returns information about support for architectural performance monitoring capabilities.

- This leaf is valid if CPUID.0AH:EAX[7:0] (Version ID) > 0 and MAX_LEAF ≥ 0AH.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-30. Leaf 0AH Architectural Performance Monitoring

Register	Field Name	Description	Domain
EAX[7:0]	VERSION	Version ID of architectural performance monitoring.	Platform
EAX[15:8]	NUM_GP_CTRS	Number of general-purpose performance monitoring counter(s) per logical processor.	Platform
EAX[23:16]	GP_CTR_WIDTH	Bit width of general-purpose, performance monitoring counter.	Platform
EAX[31:24]	EVENT_ENUM_LENGTH	Length of EBX bit vector to enumerate architectural performance monitoring events. Architectural event x is supported if EBX[x]=0 && EAX[31:24]>x.	Platform
EBX[0]	CORE_CYC_NA	Core cycle event not available if 1 or if EAX[31:24]<1.	Platform
EBX[1]	INTR_RET_NA	Instruction retired event not available if 1 or if EAX[31:24]<2.	Platform
EBX[2]	REF_CYC_NA	Reference cycles event not available if 1 or if EAX[31:24]<3.	Platform
EBX[3]	LLC_CYC_NA	Last-level cache reference event not available if 1 or if EAX[31:24]<4.	Platform
EBX[4]	LLC_MISSES_NA	Last-level cache misses event not available if 1 or if EAX[31:24]<5.	Platform
EBX[5]	BR_INSTR_RET_NA	Branch instruction retired event not available if 1 or if EAX[31:24]<6.	Platform
EBX[6]	BR_MISPRED_RET_NA	Branch mispredict retired event not available if 1 or if EAX[31:24]<7.	Platform
EBX[7]	SLOTS_NA	Top-down slots event not available if 1 or if EAX[31:24]<8.	Platform
EBX[8]	BACKEND_NA	Topdown backend bound not available if 1 or if EAX[31:24] < 9.	Platform
EBX[9]	BADSPEC_NA	Topdown bad speculation not available if 1 or if EAX[31:24] < 10.	Platform
EBX[10]	FRONTEND_NA	Topdown frontend bound not available if 1 or if EAX[31:24] < 11.	Platform
EBX[11]	RETIRING_NA	Topdown retiring not available if 1 or if EAX[31:24] < 12.	Platform
EBX[12]	LBR_INSERTS_NA	LBR inserts not available if 1 or if EAX[31:24] < 13.	Platform
EBX[31:13]	Reserved	Reserved.	

ECX[31:0]	FIXED_CTR_MASK	Supported fixed counters bit mask. Fixed-function performance counter 'i' is supported if bit 'i' is 1 (first counter index starts at zero). It is recommended to use the following logic to determine if a Fixed Counter is supported: FxCtr[i]_is_supported := ECX[i] (EDX[4:0] > i);	Platform
EDX[4:0]	NUM_FIXED_CTR	Number of contiguous fixed-function performance counters starting from 0 (if Version ID > 1).	Platform
EDX[12:5]	FIXED_CTR_WIDTH	Bit width of fixed-function performance counters (if Version ID > 1).	Platform
EDX[14:13]	Reserved	Reserved.	
EDX[15]	ANYTHREAD_DEPRECATION	Starting with Architectural Performance Monitoring Version 5, this field indicates that a processor supports AnyThread mode deprecation. If this field is set, software can choose to ignore guidelines in "AnyThread Counting and Software Evolution" of Chapter 21, "Performance Monitoring," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B	Platform
EDX[19:16]	SLOTS_PER_CYC	If this field is non-zero, it represents the number of Top-down Microarchitecture Analysis (TMA) slots per cycle. This number can be multiplied by the number of cycles (from CPU_CLK_UNHALTED.THREAD / CPU_CLK_UNHALTED.CORE or IA32_FIXED_CTR1) to determine the total number of slots. If this field is zero, IA32_FIXED_CTR3 should be used to determine the total number of slots.	Platform
EDX[31:20]	Reserved	Reserved.	

For each version of architectural performance monitoring capability, software must enumerate this leaf to discover the programming facilities and the architectural performance events available in the processor. The details are described in [Chapter 21, "Performance Monitoring," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C](#).

CPUID.0BH -- Extended Topology

CPUID.0BH returns information about Extended Topology. CPUID.1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of leaf 1FH before using leaf 0BH.

- This leaf is valid if CPUID.0BH.00H:EBX[15:0] $\neq$ 0 and MAX_LEAF $\geq$ 0BH.
 - When the leaf is invalid, CPUID.0BH.00H:ECX.DOMAIN_TYPE[15:8] will report the Domain Type ID as Invalid (0).
- The sub-leaves are enumerated until sub-leaf n returns 0 in EBX[15:0].
- If ECX contains an invalid sub-leaf index, EAX/EBX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EBX[15:0] as 0.

CPUID.0BH -- ECX $\geq$ 0

Table 21-31. Leaf 0BH Extended Topology

Register	Field Name	Description	Domain
EAX[4:0]	SHIFT_COUNT	The number of bits that the x2APIC ID must be shifted to the right to address instances of the next higher-scoped domain. When logical processor is not supported by the processor, the value of this field at the Logical Processor domain sub-leaf may be returned as either 0 (no allocated bits in the x2APIC ID) or 1 (one allocated bit in the x2APIC ID); software should plan accordingly.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[15:0]	NEXT_LEVEL_NUM_LP	The number of logical processors across all instances of this domain within the next higher-scoped domain. (For example, in a processor socket/package comprising "M" cores of "N" logical processors each, the "core" domain sub-leaf value of this field would be M*N.) This number reflects configuration as shipped by Intel. This field may also contain asymmetric values across different logical processors, as an example of a mix of cores that support more than one logical processor with cores that support only one logical processor. Note, software must not use this field to enumerate processor topology. Software must not use the value of EBX[15:0] to enumerate processor topology of the system. The value is only intended for display and diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations.	Logical Processor
EBX[31:16]	Reserved	Reserved.	
ECX[7:0]	LEVEL_NUM	The input ECX sub-leaf index.	Platform

ECX[15:8]	DOMAIN_TYPE	This field provides an identification value which indicates the domain shown in the table. Although domains are ordered, their assigned identification values are not and software should not depend on it. Note that enumeration values of 0 and 3-255 are reserved.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	X2APIC_ID	The X2APIC ID of this logical processor.	Logical Processor

The sub-leaves of CPUID.0BH describe an ordered hierarchy of logical processors starting from the smallest-scoped domain of a Logical Processor (sub-leaf index 0) to the Core domain (sub-leaf index 1) to the largest-scoped domain (the last valid sub-leaf index) that is implicitly subordinate to the unenumerated highest-scoped domain of the processor package (socket).

The details of each valid domain is enumerated by a corresponding sub-leaf. Details for a domain include its type and how all instances of that domain determine the number of logical processors and x2 APIC ID partitioning at the next higher-scoped domain. The ordering of domains within the hierarchy is fixed architecturally as shown below. For a given processor, not all domains may be relevant or enumerated; however, the logical processor and core domains are always enumerated.

For two valid sub-leaves N and N+1, sub-leaf N+1 represents the next immediate higher-scoped domain with respect to the domain of sub-leaf N for the given processor.

If sub-leaf index “N” returns an invalid domain type in ECX[15:08] (00H), then all sub-leaves with an index greater than “N” also return an invalid domain type. A sub-leaf returning an invalid domain always returns 0 in EAX and EBX.

Table 21-32. Hierarchy of Valid Domain Enumerations in CPUID.0BH:ECX[15:8]

Hierarchy	Domain	Domain Type ID Value
Invalid	Invalid	0
Lowest	Logical Processor	1
...	Core	2
Highest	Package/Socket	(Implied)
Reserved	Reserved	3-255

CPUID.0CH -- Reserved

Table 21-33. Leaf 0CH Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.0DH -- Processor Extended State

CPUID.0DH returns a bit-vector representation of all processor state extensions that are supported in the processor and storage size requirements of the XSAVE/XRSTOR area.

- This leaf is valid if CPUID.01H:ECX.XSAVE[26] = 1 and MAX_LEAF ≥ 0DH.
- Sub-leaves 0 and 1 are always valid; consult them to determine which other sub-leaves are present as described in “CPUID.0DH.n, n>01H—State Sub-Leaves”.

CPUID.0DH.00H -- Processor Extended State Main Sub-Leaf

CPUID.0DH.00H returns the processor extended state information.

Table 21-34. Leaf 0DH.00H Processor Extended State

Register	Field Name	Description	Domain
EAX[0]	X87	x87 state.	Platform
EAX[1]	SSE	SSE state.	Platform
EAX[2]	AVX	AVX state.	Platform
EAX[3]	MPX_BNDREGS	MPX state.	Platform
EAX[4]	MPX_BNDCSR	MPX state.	Platform
EAX[5]	AVX512_OPMASK	AVX-512 Opmask state.	Platform
EAX[6]	AVX512_ZMM_HI256	AVX-512 ZMM upper 256 data state.	Platform
EAX[7]	AVX512_HI16_ZMM	AVX-512 upper 16 ZMM registers state.	Platform
EAX[8]	N/A	Always returns 0 (Allocated for IA32_XSS).	Platform
EAX[9]	PKRU	PKRU state.	Platform
EAX[16:10]	N/A	Always returns 0 (Allocated for IA32_XSS).	Platform
EAX[17]	AMX_TILECFG	TILECFG state.	Platform
EAX[18]	AMX_TILEDATA	TILEDATA state.	Platform
EAX[31:19]	Reserved	Reserved.	
EBX[31:0]	XSAVE_BYTES_ENABLED_FEATURES	Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than ECX if some features at the end of the XSAVE save area are not enabled.	Logical Processor
ECX[31:0]	XSAVE_BYTES_SUPPORTED_FEATURES	Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processors, i.e. all the valid bit fields in XCR0.	Platform
EDX[31:0]	VALID_XCR0_UPPER_32	Reports the supported bits of the upper 32 bits of XCR0. XCR0[n+32] can be set to 1 only if EDX[n] is 1.	Platform

The EAX register of CPUID.0DH.00H reports the supported bits of the lower 32 bits of XCR0. XCR0[n] can be set to 1 only if EAX[n] is 1. The details are described in Chapter 13.2, “Enumeration of CPUID Support for XSAVE Instructions and XSAVE-Supported Features,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

CPUID.0DH.01H -- Feature and Supervisor State Sub-Leaf

CPUID.0DH.01H returns feature and supervisor state information.

Table 21-35. Leaf 0DH.01H Processor Extended State

Register	Field Name	Description	Domain
EAX[0]	XSAVEOPT	If 1, supports XSAVEOPT.	Platform
EAX[1]	XSAVEC	If 1, supports XSAVEC and the compacted form of XRSTOR.	Platform
EAX[2]	XGETBV1	If 1, supports XGETBV with ECX = 1.	Platform
EAX[3]	XSAVES	If 1, supports XSAVES/XRSTORS and IA32_XSS.	Platform
EAX[4]	XFD	If 1, supports extended feature disable (XFD).	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	XSAVES_BYTES_ENABLED_FEATURES	The size in bytes of the XSAVE area containing all states enabled by XCR0 IA32_XSS. If EAX[3] is enumerated as 0 and EAX[1] is enumerated as 1, EBX enumerates the size of the XSAVE area containing all states enabled by XCR0. If EAX[1] and EAX[3] are both enumerated as 0, EBX enumerates zero.	Logical Processor
ECX[7:0]	N/A	Always returns 0 (Allocated for XCR0).	Platform
ECX[8]	PT	PT state.	Platform
ECX[9]	Reserved	Always returns 0 (Allocated for XCR0).	
ECX[10]	PASID	PASID state.	Platform
ECX[11]	CET_U	CET user state.	Platform
ECX[12]	CET_S	CET supervisor state.	Platform
ECX[13]	HDC	HDC state.	Platform
ECX[14]	UINTR	UINTR state.	Platform
ECX[15]	LBR	LBR state (only for the architectural LBR feature).	Platform
ECX[16]	HWP	HWP state.	Platform
ECX[18:17]	N/A	Always returns 0 (Allocated for XCR0).	Platform
ECX[31:19]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved	

NOTE

ECX reports the supported bits of the lower 32 bits of the IA32_XSS MSR. IA32_XSS[n] can be set to 1 only if ECX[n] is 1. EDX reports the supported bits of the upper 32 bits of the IA32_XSS MSR. IA32_XSS[n+32] can be set to 1 only if EDX[n] is 1. The details are described in Chapter 13.2, "Enumeration of CPUID Support for XSAVE Instructions and XSAVE-Supported Features," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

CPUID.0DH.SUB-LEAVES -- Sub-leaves

CPUID.0DH.n, where $n > 1$, returns information about the size and offset of each processor extended state save area within the XSAVE/XRSTOR area. Software can use the forward-extendable technique depicted below to query in bulk the valid sub-leaves and obtain size and offset information for each processor extended state save area that is supported:

```

/** For each supported feature indicated by sub-leaf 0 and 1, read the size and offset sub-leaf */
For j= 2 to 62
  If (CPUID.0DH.00H:<EDX:EAX>[j] == 1 or // Use 64-bit value of EDX:EAX
    CPUID.0DH.01H:<EDX:ECX>[j] == 1) // Use 64-bit value of EDX:ECX
    Read(CPUID.0DH.j) // Examine the size and offset.
  END IF
END FOR

```

Table 21-36. Leaf 0DH.SUB-LEAVES Processor Extended State

Register	Field Name	Description	Domain
EAX[31:0]	COMP_SIZE	The size in bytes (from the offset specified in EBX) of the save area for an extended state feature associated with a valid sub-leaf index n.	Platform
EBX[31:0]	COMP_OFFSET	The offset in bytes of this extended state component's save area from the beginning of the XSAVE/XRSTOR area. This field reports 0 if the sub-leaf index, n, does not map to a valid bit in the XCR0 register. If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf n ($0 \leq n \leq 31$) is invalid if sub-leaf 0 returns 0 in EAX[n] and sub-leaf 1 returns 0 in ECX[n]. Sub-leaf n ($32 \leq n \leq 63$) is invalid if sub-leaf 0 returns 0 in EDX[n-32] and sub-leaf 1 returns 0 in EDX[n-32]	Platform
ECX[0]	COMP_SUP	This bit is set if the bit n (corresponding to the sub-leaf index) is supported in the IA32_XSS MSR; it is clear if bit n is instead supported in XCR0.	Platform
ECX[1]	COMP_64B_ALIGNED	This bit is set if, when the compacted format of an XSAVE area is used, this extended state component located on the next 64-byte boundary following the preceding state component (otherwise, it is located immediately following the preceding state component)	Platform
ECX[2]	COMP_XFD	This bit is set to indicate support for XFD faulting.	Platform
ECX[31:3]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.0EH -- Reserved

This leaf is reserved.

Table 21-37. Leaf 0EH Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.0FH -- Intel® Resource Director Technology (Intel® RDT) Monitoring

CPUID.0FH returns information for the Intel Resource Director Technology Monitoring capabilities.

As described below, software uses the bit vector returned in EDX by sub-leaf 00H to determine the available resource types (ResID) that can be monitored. This information is necessary for software to program the IA32_PQR_ASSOC and IA32_QM_EVTSEL MSRs such that Quality-of-Service data can be read afterwards from the IA32_QM_CTR MSR.

- This leaf is valid if CPUID.07H.00H:EBX.RDT_M[12] = 1 and MAX_LEAF ≥ 0FH.
- If the leaf is valid, sub-leaf 00H is always valid. Sub-leaf n (n ≥ 1) is only valid when (CPUID.0FH.00H:EDX[n] == 1).

CPUID.0FH.00H -- Intel® RDT Monitoring Main Sub-Leaf

CPUID.0FH.00H returns information about Intel RDT Monitoring.

Table 21-38. Leaf 0FH.00H Intel® Resource Director Technology (Intel® RDT) Monitoring

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	MAX_RMID	Maximum range (zero-based) of RMID within this physical processor of all types.	Platform
ECX[31:0]	Reserved	Reserved.	
EDX[0]	Reserved	Reserved.	
EDX[1]	L3_MON	If 1, supports L3 Cache Intel RDT Monitoring. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX.	Platform
EDX[31:2]	Reserved	Reserved.	

CPUID.0FH.00H returns information about the bit-vector representation of QoS monitoring resource types that are supported in the processor and maximum range of RMID values the processor can use to monitor of any supported resource types. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS monitoring capability available for that type.

CPUID.0FH.01H -- L3 Cache Intel® Resource Director Technology Monitoring

CPUID.0FH.01H returns information about L3 Cache Intel RDT monitoring.

Table 21-39. Leaf 0FH.01H Intel® Resource Director Technology (Intel® RDT) Monitoring

Register	Field Name	Description	Domain
EAX[7:0]	CTR_WIDTH	The counter width is encoded as an offset from 24b. A value of zero in this field indicates that 24-bit counters are supported. A value of 8 in this field indicates that 32-bit counters are supported.	Platform
EAX[8]	RDT_M_OVF	If 1, supports an overflow bit in the IA32_QM_CTR MSR (bit 61).	Platform
EAX[9]	IO_RDT_CMT	If 1, indicates the presence of non-CPU agent supporting Intel RDT CMT.	Platform
EAX[10]	IO_RDT_MBM	If 1, indicates the presence of non-CPU agent supporting Intel RDT MBM support.	Platform

EAX[31:11]	Reserved	Reserved.	
EBX[31:0]	CONV_FACTOR	Factor used to convert from reported IA32_QM_CTR value to derived occupancy metric (bytes) and Memory Bandwidth Monitoring (MBM) metrics.	Platform
ECX[31:0]	MAX_RMID_L3	Maximum range (zero-based) of RMID of this resource type.	Platform
EDX[0]	CMT_L3_OCCUP	If 1, supports L3 occupancy monitoring.	Platform
EDX[1]	MBM_L3_TOTAL	If 1, supports L3 total bandwidth monitoring.	Platform
EDX[2]	MBM_L3_LOCAL	If 1, supports L3 local bandwidth monitoring.	Platform
EDX[31:3]	Reserved	Reserved.	

CPUID.10H -- Intel® Resource Director Technology (Intel® RDT) Allocation

CPUID.10H returns information for Intel Resource Director Technology Allocation. This leaf is valid when CPUID.07H.00H:EBX.RDT_A[15] = 1. As described below, software uses the bit vector returned in EBX by subleaf 00H to determine the available QoS Enforcement (allocation) resource types that are supported in the processor. This information is necessary for software to configure each class of services using capability bit masks in the QoS Mask registers, IA32_resourceType_Mask_n.

- This leaf is valid if CPUID.07H.00H:EBX.RDT_A[15] = 1 and MAX_LEAF ≥ 10H.
- If the leaf is valid, sub-leaf 00H is always valid. Sub-leaf n (n ≥ 1) is only valid when (CPUID.10H.00H:EBX[n] == 1).

CPUID.10H.00H -- Intel® RDT Allocation Main Sub-Leaf

CPUID.10H.00H returns information about Intel RDT Allocation.

Table 21-40. Leaf 10H.00H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[0]	Reserved	Reserved.	
EBX[1]	CAT_L3	If 1, supports L3 Cache Allocation Technology. Sub-leaf index 0 reports valid resource identification (ResID) starting at bit position 1 of EBX.	Platform
EBX[2]	CAT_L2	If 1, supports L2 Cache Allocation Technology.	Platform
EBX[3]	MBA	If 1, supports Memory Bandwidth Allocation.	Platform
EBX[31:4]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.10H.00H returns information about the bit-vector representation of QoS Enforcement resource types that are supported in the processor. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS enforcement capability available for that type.

CPUID.10H.01H -- L3 Cache Allocation Technology

CPUID.10H.ResID=1 returns information about L3 Cache Allocation Technology.

Table 21-41. Leaf 10H.01H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[4:0]	CAT_L3_BITMASK_LENGTH	Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	CAT_L3_CONTENTION	Bit-granular map of isolation/contention of allocation units.	Platform
ECX[0]	Reserved	Reserved.	
ECX[1]	CAT_L3_NONCPU	If 1, supports L3 CAT for non-CPU agents.	Platform

ECX[2]	CAT_L3_CDP	If 1, supports L3 Code and Data Prioritization Technology.	Platform
ECX[3]	CAT_L3_NONCONTIG	If 1, supports non-contiguous capacity bitmasks. The bits that are set in the various IA32_L3_MASK_n registers do not have to be contiguous.	Platform
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CAT_L3_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.10H.02H -- L2 Cache Allocation Technology

CPUID.10H.ResID=2 returns information about L2 Cache Allocation Technology.

Table 21-42. Leaf 10H.02H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[4:0]	CAT_L2_BITMASK_LENGTH	Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	CAT_L2_CONTENTION	Bit-granular map of isolation/contention of allocation units.	Platform
ECX[1:0]	Reserved	Reserved.	
ECX[2]	CAT_L2_CDP	If 1, supports L2 Code and Data Prioritization Technology.	Platform
ECX[3]	CAT_L2_NONCONTIG	If 1, supports non-contiguous capacity bitmasks. The bits that are set in the various IA32_L2_MASK_n registers do not have to be contiguous.	Platform
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CAT_L2_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.10H.03H -- Memory Bandwidth Allocation

CPUID.10H.ResID=3 returns information about Memory Bandwidth Allocation.

Table 21-43. Leaf 10H.03H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[11:0]	MBA_MAX	Reports the maximum MBA throttling value supported for the corresponding ResID. Add one to the return value to get the result.	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[0]	PER_THREAD_MBA	Per-thread MBA controls are supported.	Platform

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

ECX[1]	Reserved	Reserved.	
ECX[2]	MBA_LINEAR	If 1, the response of the delay values is linear.	Platform
ECX[31:3]	Reserved	Reserved.	
EDX[15:0]	MBA_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.11H -- Reserved

This leaf is reserved.

Table 21-44. Leaf 11H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.12H -- Intel® Software Guard Extensions (Intel® SGX) Capability

CPUID.12H returns information about Intel® SGX capabilities. More details can be found in Chapter 35, “Introduction to Intel® Software Guard Extensions,” and Chapter 36, “Enclave Access Control and Data Structures,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D.

- This leaf is valid when CPUID.07H.00H:EBX.SGX[2] = 1 and MAX_LEAF ≥ 12H.
- If the leaf is valid, sub-leaf 00H and 01H are always valid. Sub-leaf n (n ≥ 2) is only valid when CPUID.12H.n:EAX[3:0] != 0.

CPUID.12H.00H -- Intel® SGX Main Sub-Leaf

CPUID.12H.00H returns information about Intel® SGX capabilities. It is only valid when CPUID.07H.00H:EBX.SGX = 1.

Table 21-45. Leaf 12H.00H Intel® Software Guard Extensions (Intel® SGX) Capability

Register	Field Name	Description	Domain
EAX[0]	SGX1	If 1, supports the collection of SGX1 Leaf functions.	Platform
EAX[1]	SGX2	If 1, supports the collection of SGX2 Leaf functions.	Platform
EAX[6:2]	Reserved	Reserved.	
EAX[7]	EVERIFYREPORT2	If 1, supports the ENCLU instruction Leaf EVERIFYREPORT2.	Platform
EAX[9:8]	Reserved	Reserved.	
EAX[10]	EUPDATESVN	If 1, supports the ENCLS instruction Leaf EUPDATESVN.	Platform
EAX[11]	EDECCSSA	If 1, supports the ENCLU instruction Leaf EDECCSSA.	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	MISCSELECT	Bit vector of supported extended SGX features. The definition of MISCSELECT can be found in Section 36.7.2, “SECS.MISCSELECT Field,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D.	Platform
ECX[31:0]	Reserved	Reserved.	
EDX[7:0]	MAX_ENCLAVE_SIZE_NOT_64	The maximum supported enclave size in non-64-bit mode is $2^{(EDX[7:0])}$.	Platform
EDX[15:8]	MAX_ENCLAVE_SIZE_64	The maximum supported enclave size in 64-bit mode is $2^{(EDX[15:8])}$.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.12H.01H -- Intel® SGX Attributes

CPUID.12H.01H returns information about Intel® SGX Attributes. It is only valid when CPUID.07H.00H:EBX.SGX = 1.

Table 21-46. Leaf 12H.01H Intel® Software Guard Extensions (Intel® SGX) Capability

Register	Field Name	Description	Domain

EAX[31:0]	ECREATE_SECS_ATTRIBUTES_31_0	Reports the valid bits of SECS.ATTRIBUTES[31:0] that software can set with ECREATE.	Platform
EBX[31:0]	ECREATE_SECS_ATTRIBUTES_63_32	Reports the valid bits of SECS.ATTRIBUTES[63:32] that software can set with ECREATE.	Platform
ECX[31:0]	ECREATE_SECS_ATTRIBUTES_95_64	Reports the valid bits of SECS.ATTRIBUTES[95:64] that software can set with ECREATE.	Platform
EDX[31:0]	ECREATE_SECS_ATTRIBUTES_127_96	Reports the valid bits of SECS.ATTRIBUTES[127:96] that software can set with ECREATE.	Platform

The definition of the attributes can be found in Section 36.7.1, "ATTRIBUTES," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D

CPUID.12H -- $n \geq 2$ - Intel® SGX Enclave Page Cache

CPUID.12H with $ECX \geq 2$ returns information about Intel® SGX Enclave Page Cache and is supported if CPUID.07H.00H:EBX.SGX = 1.

For sub-leaves where $ECX \geq 2$, the definition of EAX[31:4], EBX, ECX, and EDX depends on the sub-leaf type listed below.

Sub-Leaf Encoding Type EAX[3:0] = 0000b (Invalid)

This sub-leaf is invalid. EDX:ECX:EBX:EAX return 0.

CPUID.12H -- Sub-Leaf Encoding Type EAX[3:0] = 0001b

This sub-leaf enumerates an EPC section with EDX:ECX, EBX:EAX defined as follows.

Table 21-47. Leaf 12H.SUB-LEAF ENCODING TYPE EAX[3:0] = 0001B Intel® Software Guard Extensions (Intel® SGX) Capability

Register	Field Name	Description	Domain
EAX[3:0]	SUB_LEAF_TYPE	Value is 0001b.	Platform
EAX[11:4]	Reserved	Reserved.	
EAX[31:12]	EPC_SECTION_ADDR_31_12	Bits 31:12 of the physical address of the base of the EPC section.	Platform
EBX[19:0]	EPC_SECTION_ADDR_51_32	Bits 51:32 of the physical address of the base of the EPC section.	Platform
EBX[31:20]	Reserved	Reserved.	

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

ECX[3:0]	EPC_SECTION_PROPERTY	EPC Section Property Encoding Definitions, as follows: 0000b – All bits in EDX:ECX are enumerated as 0. 0001b – This section has confidentiality, integrity, and replay protection. 0010b – This section has confidentiality protection only. 0011b – This section has confidentiality and integrity protection. All other encodings are reserved.	Platform
ECX[11:4]	Reserved	Reserved.	
ECX[31:12]	EPC_SECTION_SIZE_31_12	Bits 31:12 of the size of the corresponding EPC section within the Processor Reserved Memory.	Platform
EDX[19:0]	EPC_SECTION_SIZE_51_32	Bits 51:32 of the size of the corresponding EPC section within the Processor Reserved Memory.	Platform
EDX[31:20]	Reserved	Reserved.	

CPUID.13H -- Reserved

This leaf is reserved.

Table 21-48. Leaf 13H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.14H -- Intel® Processor Trace (Intel® PT)

CPUID.14H returns information about Intel® Processor Trace (PT).

CPUID.14H.00H returns information about Intel Processor Trace extensions.

CPUID.14H.n (n > 0 and less than the number of non-zero bits in CPUID.14H.00H:EAX) returns information about packet generation in Intel Processor Trace.

For more details on Intel PT, see Chapter 34, “Intel® Processor Trace,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D.

- This leaf is valid when CPUID.07H.00H:EBX.INTEL_PROC_TRACE[25] = 1 and MAX_LEAF ≥ 14H.
- The maximum sub-leaf value for ECX is specified in CPUID.14H.00H:EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.14H.00H -- Intel® PT Main Sub-Leaf

CPUID.14H.00H returns information about Intel Processor Trace extensions.

Table 21-49. Leaf 14H.00H Intel® Processor Trace (Intel® PT)

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum sub-leaf supported in leaf 14H.	Platform
EBX[0]	CR3_FILTER	If 1, supports that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed.	Platform
EBX[1]	CYC_ACC	If 1, supports Configurable PSB and Cycle-Accurate Mode.	Platform
EBX[2]	IP_FILTER	If 1, supports IP Filtering, TraceStop filtering, and preservation of Intel PT MSRs across warm reset.	Platform
EBX[3]	MTC	If 1, supports MTC timing packet and suppression of COFI-based packets.	Platform
EBX[4]	PTWRITE	If 1, supports PTWRITE. Writes can set IA32_RTIT_CTL[12] (PTWEn) and IA32_RTIT_CTL[5] (FUPonPTw), and PTWRITE can generate packets.	Platform
EBX[5]	PWR_EVT_TRACE	If 1, supports Power Event Trace. Writes can set IA32_RTIT_CTL[4] (PwrEvtEn), enabling Power Event Trace packet generation.	Platform
EBX[6]	PMI_PRESERVE	If 1, supports the PSB and PMI preservation. Writes can set IA32_RTIT_CTL[56] (InjectPsbPmiOnEn-able), enabling the processor to set IA32_RTIT_STATUS[7] (PendToPaPMI) and/or IA32_RTIT_STATUS[6] (PendPSB) in order to preserve ToPA PMIs and/or PSBs otherwise lost due to Intel PT disable. Writes can also set PendToPAPMI and PendPSB.	Platform
EBX[7]	EVENT_TRACE	If 1, supports that writes can set IA32_RTIT_CTL[31] (EventEn), enabling Event Trace packet generation.	Platform
EBX[8]	TNT_DIS	If 1, supports that writes can set IA32_RTIT_CTL[55] (DisTNT), disabling TNT packet generation.	Platform

EBX[9]	PTTT	If 1, Processor Trace Trigger Tracing (PTTT) is supported.	Platform
EBX[31:10]	Reserved	Reserved.	
ECX[0]	TOPAOUT	If 1, supports that tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme; IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUPUT_MASK_PTRS MSRs can be accessed.	Platform
ECX[1]	MENTRY	If 1, supports that ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOrTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS.	Platform
ECX[2]	SNGL_RNG_OUT	If 1, supports the Single-Range Output scheme.	Platform
ECX[3]	TRACE_TRANSPORT_SUBSYSTEM	If 1, supports the output to Trace Transport subsystem.	Platform
ECX[30:4]	Reserved	Reserved.	
ECX[31]	LIP	If 1, the generated packets which contain IP payloads contain LIP. If 0, the generated packets which contain IP payloads contain Effective IP. Trace segments using a flat memory model will generate the same information regardless of how a logical processor reports this value since LIP=EIP.	Logical Processor
EDX[31:0]	Reserved	Reserved.	

CPUID.14H.01H -- Feature Information Sub-Leaf

CPUID.14H.01H returns information about packet generation in Intel Processor Trace.

Table 21-50. Leaf 14H.01H Intel® Processor Trace (Intel® PT)

Register	Field Name	Description	Domain
EAX[2:0]	RANGECNT	Number of configurable Address Ranges for filtering.	Platform
EAX[7:3]	Reserved	Reserved.	
EAX[10:8]	TRIGGER_CFG_CNT	Number of IA32_RTIT_TRIGGERx_CFG MSRs. The number of triggers supported is 4x this value.	Platform
EAX[15:11]	Reserved	Reserved.	
EAX[31:16]	MTC_RATE	Bitmap of supported MTC period encodings.	Platform
EBX[15:0]	CYC_THRESHOLDS	Bitmap of supported Cycle Threshold value encodings.	Platform
EBX[31:16]	PSB_RATE	Bitmap of supported Configurable PSB frequency encodings.	Platform
ECX[0]	ICNT	If 1, the trigger action EN_ICNT is supported.	Platform
ECX[1]	TRIGGER_PAUSE	If 1, the trigger actions TRACE_PAUSE and TRACE_RESUME are supported.	Platform
ECX[14:2]	Reserved	Reserved.	

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

ECX[15]	TRIGGER_DR_MATCH	If 1, trigger input DR match is supported.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.15H -- Time Stamp Counter and Nominal Core Crystal Clock

CPUID.15H returns information about the Time Stamp Counter and the Nominal Core Crystal Clock.

- This leaf is valid if MAX_LEAF $\geq$ 15H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-51. Leaf 15H Time Stamp Counter and Nominal Core Crystal Clock

Register	Field Name	Description	Domain
EAX[31:0]	DENOMINATOR	An unsigned integer which is the denominator of the TSC/"core crystal clock" ratio.	Platform
EBX[31:0]	NUMERATOR	An unsigned integer which is the numerator of the TSC/"core crystal clock" ratio. If 0, the TSC/"core crystal clock" ratio is not enumerated.	Platform
ECX[31:0]	NOMINAL_ART_FREQUENCY	An unsigned integer which is the nominal frequency of the core crystal clock in Hz. If 0, the nominal core crystal clock frequency is not enumerated. Note, the core crystal clock may differ from the reference clock, bus clock or core clock frequencies.	Platform
EDX[31:0]	Reserved	Reserved.	

Dividing EBX[31:0] by EAX[31:0] provides the ratio of the TSC frequency to the core crystal clock frequency. The Timestamp Counter frequency is computed by multiplying that ratio by ECX[31:0], such as $TSC\_frequency = ECX * EBX/EAX$.

CPUID.16H -- Processor Frequency Information

CPUID.16H returns information about processor frequency information.

Data is returned from this interface in accordance with the processor's specification and does not reflect actual values. Suitable use of this data includes the display of processor information in like manner to the processor brand string and for determining the appropriate range to use when displaying processor information e.g. frequency history graphs. The returned information should not be used for any other purpose as the returned information does not accurately correlate to information / counters returned by other processor interfaces.

While a processor may support the Processor Frequency Information leaf, fields that return a value of zero are not supported.

- This leaf is valid if MAX_LEAF $\geq$ 16H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-52. Leaf 16H Processor Frequency Information

Register	Field Name	Description	Domain
EAX[15:0]	PROCESSOR_BASE_FREQUENCY	Processor Base Frequency (in MHz).	Logical Processor
EAX[31:16]	Reserved	Reserved.	
EBX[15:0]	MAXIMUM_FREQUENCY	Maximum Frequency (in MHz).	Logical Processor
EBX[31:16]	Reserved	Reserved.	
ECX[15:0]	BUS_FREQUENCY	Bus (Reference) Frequency (in MHz).	Logical Processor
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.17H -- System-on-Chip Vendor Attribute

CPUID.17H returns System-on-Chip vendor attribute information.

- This leaf is valid if CPUID.17H.00H:EAX[31:0] (MaxSOCID_Index) ≥ 3 and MAX_LEAF $\geq 17H$.
- The maximum sub-leaf value for ECX is specified in CPUID.17H.00H:EAX[31:0] MaxSOCID_Index.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.17H.00H -- Main Sub-Leaf

CPUID.17H.00H returns System-on-Chip vendor attribute information.

Table 21-53. Leaf 17H.00H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SOCID_INDEX	Reports the maximum input value of supported Sub-leaf in Leaf 17H.	Platform
EBX[15:0]	SOC_VENDOR_ID	SOC Vendor ID.	Platform
EBX[16]	IS_VENDOR_SCHEME	If 1, the SOC Vendor ID field is assigned via an industry standard enumeration scheme. Otherwise, the SOC Vendor ID field is assigned by Intel.	Platform
EBX[31:17]	Reserved	Reserved.	
ECX[31:0]	PROJECT_ID	A unique number an SOC vendor assigns to its SOC projects.	Platform
EDX[31:0]	STEPPING_ID	A unique number within an SOC project that an SOC vendor assigns.	Package

CPUID.17H.01H -- Vendor Brand String Sub-Leaf (Bytes 0 to 15)

CPUID.17H with ECX=1, 2, or 3 returns information for System-on-Chip vendor branding string.

SOC Vendor Brand String is a UTF-8 encoded string padded with trailing bytes of 00H. The complete SOC Vendor Brand String is constructed by concatenating in ascending order the output of each sub-leaf 1 to 3. For the output of each sub-leaf, byte are ordered in ascending order of EAX:EBX:ECX:EDX.

Leaf 17H sub-leaves 4 and above are reserved.

Table 21-54. Leaf 17H.01H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	VENDOR_BRAND_STRING_BYTES_0_to_3	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EBX[31:0]	VENDOR_BRAND_STRING_BYTES_4_to_7	SOC Vendor Brand String. UTF-8 encoded string.	Platform
ECX[31:0]	VENDOR_BRAND_STRING_BYTES_8_to_11	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EDX[31:0]	VENDOR_BRAND_STRING_BYTES_12_to_15	SOC Vendor Brand String. UTF-8 encoded string.	Platform

CPUID.17H.02H -- Vendor Brand String Sub-Leaf (Bytes 16 to 31)

Table 21-55. Leaf 17H.02H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	VENDOR_BRAND_STRING_BYTES_16_to_19	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EBX[31:0]	VENDOR_BRAND_STRING_BYTES_20_to_23	SOC Vendor Brand String. UTF-8 encoded string.	Platform

ECX[31:0]	VENDOR_BRAND_STRING_BYTES_24_to_27	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EDX[31:0]	VENDOR_BRAND_STRING_BYTES_28_to_31	SOC Vendor Brand String. UTF-8 encoded string.	Platform

CPUID.17H.03H -- Vendor Brand String Sub-Leaf (Bytes 32 to 47)

Table 21-56. Leaf 17H.03H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	VENDOR_BRAND_STRING_BYTES_32_to_35	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EBX[31:0]	VENDOR_BRAND_STRING_BYTES_36_to_39	SOC Vendor Brand String. UTF-8 encoded string.	Platform
ECX[31:0]	VENDOR_BRAND_STRING_BYTES_40_to_43	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EDX[31:0]	VENDOR_BRAND_STRING_BYTES_44_to_47	SOC Vendor Brand String. UTF-8 encoded string.	Platform

CPUID.17H.M>MAXSOCID_INDEX—RESERVED SUB-LEAVES -- m>MaxSOCID_Index—Reserved Sub-Leaves

CPUID.17H with ECX > MaxSOCID_Index is reserved and returns all zeroes.

Table 21-57. Leaf 17H.M>MAXSOCID_INDEX—RESERVED SUB-LEAVES System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved	
EBX[31:0]	Reserved	Reserved	
ECX[31:0]	Reserved	Reserved	
EDX[31:0]	Reserved	Reserved	

CPUID.18H -- Deterministic Address Translation Parameters

CPUID.18H returns information about the Deterministic Address Translation Parameters. Each sub-leaf enumerates a different address translation structure.

- This leaf is valid if CPUID.18H.00H:EAX[31:0] $\neq 0$ and MAX_LEAF $\geq 18H$.
- The maximum sub-leaf value for ECX is specified in CPUID.18H.00H:EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX. A sub-leaf index is also invalid if EDX[4:0] returns 0.
- Valid sub-leaves do not need to be contiguous or in any particular order. A valid sub-leaf may be in a higher input ECX value than an invalid sub-leaf or than a valid sub-leaf of a higher or lower-level structure.

CPUID.18H.00H -- Main Sub-Leaf

Deterministic Address Translation Parameters Main Leaf

Table 21-58. Leaf 18H.00H Deterministic Address Translation Parameters

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum input value of supported sub-leaf in leaf 18H.	Logical Processor
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[4:0]	TYPE	Will always return 0.	Logical Processor
EDX[31:5]	Reserved	Reserved.	

CPUID.18H.ECX ≥ 1 -- Sub-Leaves

CPUID.18H with ECX ≥ 1 returns Deterministic Address Translation Parameters information.

Table 21-59. Leaf 18H.ECX ≥ 1 Deterministic Address Translation Parameters

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[0]	4KB_ENTRIES	If 1, supports 4K page size entries in this structure.	Logical Processor
EBX[1]	2MB_ENTRIES	If 1, supports 2MB page size entries in this structure.	Logical Processor
EBX[2]	4MB_ENTRIES	If 1, supports 4MB page size entries in this structure.	Logical Processor
EBX[3]	1GB_ENTRIES	If 1, supports 1 GB page size entries in this structure.	Logical Processor
EBX[7:4]	Reserved	Reserved.	
EBX[10:8]	PARTITIONING	Partitioning (0:Soft partitioning between the logical processors sharing this structure).	Logical Processor
EBX[15:11]	Reserved	Reserved.	
EBX[31:16]	NUM_WAYS	W = ways of associativity.	Logical Processor
ECX[31:0]	NUM_SETS	S = number of sets.	Logical Processor

EDX[4:0]	TYPE	00000b: Null (indicates this Sub-leaf is not valid). 00001b: Data TLB. 00010b: Instruction TLB. 00011b: Unified TLB.1 00100b: Load Only TLB. Hit on loads; fills on both loads and stores. 00101b: Store Only TLB. Hit on stores; fill on stores. All other encodings are reserved. Some unified TLBs will allow a single TLB entry to satisfy data read/write and instruction fetches. Others will require separate entries (e.g., one loaded on data read/write and another loaded on an instruction fetch). See the Intel® 64 and IA-32 Architectures Optimization Reference Manual for details of a particular product.	Logical Processor
EDX[7:5]	LEVEL_NUM	Translation cache level (starts at 1).	Logical Processor
EDX[8]	FULLY_ASSOC	Fully associative structure.	Logical Processor
EDX[13:9]	Reserved	Reserved.	
EDX[25:14]	MAX_LP_ADDRESSABLE_IDS	Maximum number of addressable IDs for logical processors sharing this translation cache. Add one to the return value to get the result.	Logical Processor
EDX[31:26]	Reserved	Reserved.	

CPUID.19H -- Key Locker

CPUID.19H returns Key Locker information.

- This leaf is valid if CPUID.07H.00H:ECX.KEY_LOCKER[23] = 1 and MAX_LEAF ≥ 19H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-60. Leaf 19H Key Locker

Register	Field Name	Description	Domain
EAX[0]	CPLD_RESTRICT	If 1, supports the Key Locker restriction of CPLD-only. ¹	Platform
EAX[1]	NO_ENCRYPT_RESTRICT	If 1, supports the Key Locker restriction of no-encrypt. ¹	Platform
EAX[2]	NO_DECRYPT_RESTRICT	If 1, supports the Key Locker restriction of no-decrypt. ¹	Platform
EAX[31:3]	Reserved	Reserved.	
EBX[0]	AESKLE	If 1, the AES Key Locker instructions are fully enabled. CPUID.19H:EBX.AESKLE[0] is enumerated as 1 if the AES Key Locker instructions have been activated by system firmware and CR4.KL[bit 19] = 1. Software can check this bit after setting CR4.KL to determine whether AES Key Locker instructions have been enabled. Note that some processors may allow enabling of those instructions without activation by system firmware. Some processors may not support the use of AES Key Locker instructions in system-management-mode (SMM). Those processors enumerate CPUID.19H:EBX.AESKLE[0] as 0 in SMM regardless of the setting of CR4.KL.	Logical Processor
EBX[1]	Reserved	Reserved.	
EBX[2]	AES_WIDE	If 1, supports the AES wide Key Locker instructions. ¹	Platform
EBX[3]	Reserved	Reserved.	
EBX[4]	IWKEYBACKUP	If 1, supports the Key Locker MSRs (IA32_COPY_LOCAL_TO_PLATFORM, IA23_COPY_PLATFORM_TO_LOCAL, IA32_COPY_STATUS, and IA32_IWKEYBACKUP_STATUS) and backing up the internal wrapping key. ¹	Platform
EBX[31:5]	Reserved	Reserved.	
ECX[0]	NOBACKUP	If 1, supports the NoBackup parameter to LOADIWKEY. ¹	Platform
ECX[1]	RAND_IWKEY	If 1, supports KeySource encoding of 1 (randomization of the internal wrapping key). ¹	Platform
ECX[31:2]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

NOTE

1. This field is valid only if CPUID.19H:EBX.AESKLE[0] = 1.

CPUID.1AH -- Native Model ID Enumeration

CPUID.1AH returns Native Model ID information.
This leaf exists on all logical processors in a hybrid package, it may also be present in other processor configurations.

- This leaf is valid if CPUID.1AH.00H:EAX[31:0] <> 0 and MAX_LEAF ≥ 1AH.
- The only valid sub-leaf is 0 and ECX must be set to 0.

Table 21-61. Leaf 1AH Native Model ID Enumeration

Register	Field Name	Description	Domain
EAX[23:0]	CORE_NATIVE_MODEL_ID	The core-type and native model ID can be used to uniquely identify the microarchitecture of the core. This native model ID is not unique across core types, and not related to the model ID reported in CPUID.01H, and does not identify the SOC.	Logical Processor
EAX[31:24]	CORE_TYPE	10H: Reserved 20H: Intel® Atom® 30H: Reserved 40H: Intel® Core The core type may only be used as an identification of the microarchitecture for this logical processor and its numeric value has no significance, neither large nor small. This field neither implies nor expresses any other attribute to this logical processor and software should not assume any.	Logical Processor
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1BH -- PCONFIG Information

CPUID.1BH -- Output Registers Format for All Sub-Leaves

CPUID.1BH returns information for PCONFIG capabilities. This information is enumerated in sub-leaves selected by the value of ECX (starting with 0).

- This leaf is valid if CPUID.07H.00H:EDX.PCONFIG[18] = 1 and MAX_LEAF ≥ 1BH.
- Sub-leaves are enumerated until sub-leaf n, where EAX[11:0] returns 0.

Table 21-62. Leaf 1BH PCONFIG Information

Register	Field Name	Description	Domain
EAX[11:0]	SUB_LEAF_TYPE	0 (Invalid)	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

Each sub-leaf of CPUID.1BH enumerates its sub-leaf type in EAX. If a sub-leaf type is 0, the sub-leaf is invalid and zero is returned in EBX, ECX, and EDX. In this case, all subsequent sub-leaves (selected by larger input values of ECX) are also invalid.

The only valid sub-leaf type currently defined is 1, indicating that the sub-leaf enumerates target identifiers for the PCONFIG instruction. Any non-zero value returned in EBX, ECX, or EDX indicates a valid target identifier of the PCONFIG instruction (any value of zero should be ignored). The only target identifier currently defined is 1, indicating TME-MK. See the “PCONFIG—Platform Configuration” instruction in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B, for more information.

CPUID.1BH.OUTPUT REGISTERS FOR SUB-LEAVE TYPE TARGET IDENTIFIER (1) -- Output Registers for Sub-Leave Type Target Identifier (1)

Table 21-63. Leaf 1BH.OUTPUT REGISTERS FOR SUB-LEAVE TYPE TARGET IDENTIFIER (1) PCONFIG Information

Register	Field Name	Description	Domain
EAX[11:0]	SUB_LEAF_TYPE	1 (Target Identifier)	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	TARGET_IDENTIFIER_1	Target identifier	Platform
ECX[31:0]	TARGET_IDENTIFIER_2	Target identifier	Platform
EDX[31:0]	TARGET_IDENTIFIER_3	Target identifier	Platform

The only current valid target identifier is 1 for MK-TME.

CPUID.1CH -- Last Branch Records (LBR) Information

CPUID.1CH returns information about architectural Last Branch Records (LBR). For details on LBR, see Chapter 20, “Last Branch Records,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B.

- This leaf is valid if CPUID.07H.00H:EDX.ARCH_LBRS[19] = 1 and MAX_LEAF ≥ 1CH.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-64. Leaf 1CH Last Branch Records (LBR) Information

Register	Field Name	Description	Domain
EAX[7:0]	LBR_DEPTH_VALUES	For each bit <i>n</i> set in this field, the IA32_LBR_DEPTH.DEPTH value $8 \cdot (n+1)$ is supported.	Platform
EAX[29:8]	Reserved	Reserved.	
EAX[30]	DEEP_C_STATE_RESET	If 1, supports that LBRs may be cleared on an MWAIT that requests a C-state numerically greater than C1.	Platform
EAX[31]	IP_VALUES_CONTAIN_LIP	If 1, the LBR IP values contain LIP. If 0, IP values contain Effective IP. Trace segments using a flat memory model will generate the same information regardless of how a logical processor reports this value since LIP = EIP	Logical Processor
EBX[0]	CPL_FILTERING	If 1, supports setting IA32_LBR_CTL[2:1] to a non-zero value.	Platform
EBX[1]	BRANCH_FILTERING	If 1, supports setting IA32_LBR_CTL[22:16] to a non-zero value.	Platform
EBX[2]	CALL_STACK_MODE	If 1, supports setting IA32_LBR_CTL[3] to 1.	Platform
EBX[31:3]	Reserved	Reserved.	
ECX[0]	MISPREDICT_BIT	If 1, IA32_LBR_x_INFO[63] holds indication of branch misprediction (MISPRED).	Platform
ECX[1]	TIMED_LBRS	If 1, IA32_LBR_x_INFO[15:0] holds CPU cycles since last LBR entry (CYC_CNT), and IA32_LBR_x_INFO[60] holds an indication of whether the value held there is valid (CYC_CNT_VALID).	Platform
ECX[2]	BRANCH_TYPE_FIELD_SUPPORTED	If 1, IA32_LBR_INFO_x[59:56] holds indication of the recorded operation’s branch type (BR_TYPE).	Platform
ECX[15:3]	Reserved	Reserved.	
ECX[19:16]	EVENT_LOGGING_BITMAP	The event logging bitmap, wherein each set bit corresponds to a programmable performance monitoring counter that supports LBR event logging.	Platform
ECX[31:20]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1DH -- Tile Information

CPUID.1DH returns information about tile architecture and tile palette 1 (see Chapter 19, “Programming with Intel® Advanced Matrix Extensions,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).

- This leaf is valid if CPUID.07H.00H:EDX.AMX_TILE[24] = 1 and MAX_LEAF ≥ 1DH.
- The maximum sub-leaf value for ECX is specified in CPUID.1DH.00H:EAX[31:0] max_palette.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.1DH.00H -- Tile Information Main Sub-Leaf

CPUID.1DH.00H returns the tile architecture information.

Table 21-65. Leaf 1DH.00H Tile Information

Register	Field Name	Description	Domain
EAX[31:0]	MAX_PALETTE	Highest numbered palette sub-leaf. Value = 1.	Platform
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1DH.01H -- Tile Palette 1

CPUID.1DH.01H returns tile palette information.

Table 21-66. Leaf 1DH.01H Tile Information

Register	Field Name	Description	Domain
EAX[15:0]	TOTAL_TILE_BYTES	Palette 1 total_tile_bytes. Value = 8192.	Platform
EAX[31:16]	BYTES_PER_TILE	Palette 1 bytes_per_tile. Value = 1024.	Platform
EBX[15:0]	BYTES_PER_ROW	Palette 1 bytes_per_row. Value = 64.	Platform
EBX[31:16]	MAX_NAMES	Palette 1 max_names (number of tile registers). Value = 8.	Platform
ECX[15:0]	MAX_ROWS	Palette 1 max_rows. Value = 16.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1EH -- TMUL Information

CPUID.1EH returns information about TMUL capabilities (see Chapter 19, “Programming with Intel® Advanced Matrix Extensions,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).

- This leaf is valid if CPUID.07H.00H:EDX.AMX_TILE[24] = 1 and MAX_LEAF ≥ 1EH.
- The only valid sub-leaf is 0 and ECX must be set to 0.

CPUID.1EH.00H -- TMUL Information Main Leaf

TMUL Main Leaf Information

Table 21-67. Leaf 1EH.00H TMUL Information

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[7:0]	TMUL_MAXK	tmul_maxk (rows or columns). Value = 16.	Platform
EBX[23:8]	TMUL_MAXN	tmul_maxn (column bytes). Value = 64.	Platform
EBX[31:24]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1FH -- V2 Extended Topology

CPUID.1FH returns information about V2 Extended Topology.

CPUID.1FH is a preferred superset to leaf 0BH. Intel recommends using leaf 1FH when available rather than leaf 0BH and ensuring that any leaf 0BH algorithms are updated to support leaf 1FH.

- This leaf is valid if CPUID.1FH.00H:EBX[15:0] $\neq$ 0 and MAX_LEAF $\geq$ 1FH.
 - When the leaf is invalid, CPUID.1FH.00H:ECX.DOMAIN_TYPE[15:8] will report the Domain Type ID as Invalid (0).
- The sub-leaves are enumerated until sub-leaf n returns 0 in EBX[15:0].
- If ECX contains an invalid sub-leaf index, EAX/EBX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EBX[15:0] as 0.

CPUID.1FH -- ECX $\geq$ 0

Table 21-68. Leaf 1FH V2 Extended Topology

Register	Field Name	Description	Domain
EAX[4:0]	SHIFT_COUNT	The number of bits that the x2APIC ID must be shifted to the right to address instances of the next higher-scoped domain. When logical processor is not supported by the processor, the value of this field at the Logical Processor domain sub-leaf may be returned as either 0 (no allocated bits in the x2APIC ID) or 1 (one allocated bit in the x2APIC ID); software should plan accordingly.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[15:0]	NEXT_LEVEL_NUM_LP	The number of logical processors across all instances of this domain within the next higher-scoped domain relative to this current logical processor. (For example, in a processor socket/package comprising “M” dies of “N” cores each, where each core has “L” logical processors, the “die” domain sub-leaf value of this field would be M*N*L. In an asymmetric topology this would be the summation of the value across the lower domain level instances to create each upper domain level instance.) This number reflects configuration as shipped by Intel. Note that the number of logical processors can be asymmetric in which case “L” may be different on different logical processors, as an example a core with 2 logical processors on the same platform as a core with 1 logical processor. Note, software must not use this field to enumerate processor topology. Software must not use the value of EBX[15:0] to enumerate processor topology of the system. The value is only intended for display and diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations.	Logical Processor
EBX[31:16]	Reserved	Reserved.	

ECX[7:0]	LEVEL_NUM	The input ECX sub-leaf index.	Platform
ECX[15:8]	DOMAIN_TYPE	This field provides an identification value which indicates the domain as shown in the table. Although domains are ordered, their assigned identification values are not and software should not depend on it. (For example, if a new domain between core and module is specified, it will have an identification value higher than 5.) See the table below for the current list of valid enumerations. Note that enumeration values of 0 and 7-255 are reserved.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	X2APIC_ID	The x2APIC ID of the current logical processor is always valid and does not vary with the sub-leaf index in ECX.	Logical Processor

The sub-leaves of CPUID.1FH describe an ordered hierarchy of logical processors starting from the smallest scoped domain of a Logical Processor (sub-leaf index 0) to the Core domain (sub-leaf index 1) to the largest scoped domain (the last valid sub-leaf index) that is implicitly subordinate to the unenumerated highest-scoped domain of the processor package (socket).

The details of each valid domain is enumerated by a corresponding sub-leaf. Details for a domain include its type and how all instances of that domain determine the number of logical processors and x2 APIC ID partitioning at the next higher-scoped domain. The ordering of domains within the hierarchy is fixed architecturally as shown below. For a given processor, not all domains may be relevant or enumerated; however, the logical processor and core domains are always enumerated. As an example, a processor may report an ordered hierarchy consisting only of "Logical Processor," "Core," and "Die."

For two valid sub-leaves N and N+1, sub-leaf N+1 represents the next immediate higher-scoped domain with respect to the domain of sub-leaf N for the given processor.

If sub-leaf index "N" returns an invalid domain type in ECX[15:08] (00H), then all sub-leaves with an index greater than "N" also return an invalid domain type. A sub-leaf returning an invalid domain always returns 0 in EAX and EBX.

Table 21-69. Hierarchy of Valid Domain Enumerations in CPUID.1FH:ECX[15:8]

Hierarchy	Domain	Domain Type ID Value
Invalid	Invalid	0
Lowest	Logical Processor	1
...	Core	2
...	Module	3
...	Tile	4
...	Die	5
...	DieGrp	6
Highest	Package/Socket	(Implied)
Reserved	Reserved	7-255

CPUID.20H -- Processor History Reset Information

CPUID.20H returns information about processor history reset when CPUID.07H.01H:EAX.HRESET[22] = 1.

- This leaf is valid if CPUID.07H.01H:EAX.HRESET[22] = 1 and MAX_LEAF ≥ 20H.
- The maximum sub-leaf value for ECX is specified in CPUID.20H.00H:EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.20H.00H -- Processor History Reset Sub-leaf

Table 21-70. Leaf 20H.00H Processor History Reset Information

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum number of sub-leaves that are supported in leaf 20H.	Platform
EBX[0]	THREAD_DIRECTOR_HRESET	Indicates support for both HRESET's EAX[0] parameter, and IA32_HRESET_ENABLE[0] set by the OS to enable reset of Intel® Thread Director history.	Platform
EBX[31:1]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.21H -- Unimplemented

Does not return feature information for the processor. Allocated for use by TDX modules; see Intel® Trust Domain Extensions (Intel® TDX) Module Base Architecture Specification. Software emulating CPUID should not change the information returned for this leaf.

CPUID.22H -- Reserved

This leaf is reserved.

Table 21-71. Leaf 22H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H -- Architectural Performance Monitoring Extended

CPUID.23H returns architectural performance monitoring extended information.

- This leaf is valid if CPUID.07H.01H:EAX.ARCH_PERFMON_EXT[8] = 1 and MAX_LEAF ≥ 23H.
- The sub-leaves of this leaf are enumerated by a bitmask specified in CPUID.23H.00H:EAX[31:0] **SUBLEAF_MASK**. The bit numbers of set bits in the bitmask represent valid sub-leaf indexes.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index is invalid if the index as a bit number is clear in the Available Sub-Leaf Mask or is greater than 31.

CPUID.23H.00H -- Main Sub-Leaf

Table 21-72. Leaf 23H.00H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	SUBLEAF_MASK	If bit n is set, sub-leaf n is supported. (For unsupported sub-leaves, 0 is returned in the registers EAX, EBX, ECX, and EDX.)	Logical Processor
EBX[0]	UNITMASK2	If 1, supports the UnitMask2 field in the IA32_PERFEVTSELx MSRs.	Logical Processor
EBX[1]	EQ	If 1, supports the equal flag in the IA32_PERFEVTSELx MSRS.	Logical Processor
EBX[31:2]	Reserved	Reserved.	
ECX[7:0]	SLOTS_PER_CYC	If this field is non-zero, it represents the number of Top-down Microarchitecture Analysis (TMA) slots per cycle. This number can be multiplied by the number of cycles (from CPU_CLK_UNHALTED.THREAD / CPU_CLK_UNHALTED.CORE or IA32_FIXED_CTR1) to determine the total number of slots. If this field is zero, IA32_FIXED_CTR3 should be used to determine the total number of slots.	Logical Processor
ECX[31:8]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H.01H -- Counter Information Sub-Leaf

Table 21-73. Leaf 23H.01H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	GP_COUNTERS	For each bit n set in this field, the processor supports general-purpose performance monitoring counter n.	Logical Processor
EBX[31:0]	FIXED_COUNTERS	For each bit m set in this field, the processor supports fixed-function performance monitoring counter m. The valid range of fixed-function counters is 0 through 15.	Logical Processor
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H.02H -- Bitmap of Auto Counter Reload Sub-Leaf

Table 21-74. Leaf 23H.02H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	ACR_GP_RELOAD	General counters that can be reloaded. For each bit n set in this field, the processor supports ACR for general-purpose performance monitoring counter n.	Logical Processor
EBX[31:0]	ACR_FIXED_RELOAD	Fixed counters that can be reloaded. For each bit m set in this field, the processor supports ACR for fixed-function performance monitoring counter m.	Logical Processor
ECX[31:0]	ACR_GP_TRIGGER	General counters that can cause reloads. For each bit y set in this field, the processor allows general-purpose performance monitoring counter y to reload all existing general-purpose performance monitoring counters capable of being reloaded.	Logical Processor
EDX[31:0]	ACR_FIXED_TRIGGER	Fixed counters that can cause reloads. For each bit x set in this field, the processor allows fixed-function performance monitoring counter x to reload all existing fixed-function performance monitoring counters capable of being reloaded.	Logical Processor

CPUID.23H.03H -- Architectural Performance Monitoring Events Bitmap Sub-Leaf

For each bit n set in this field, the processor supports Architectural Performance Monitoring Event of index n.

Table 21-75. Leaf 23H.03H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[0]	CORE_CYC	If 1, supports architectural index 0.	Logical Processor
EAX[1]	INSTR_RET	If 1, supports architectural index 1.	Logical Processor
EAX[2]	REF_CYC	If 1, supports architectural index 2.	Logical Processor
EAX[3]	LLC_REF	If 1, supports architectural index 3.	Logical Processor
EAX[4]	LLC_MISSES	If 1, supports architectural index 4.	Logical Processor
EAX[5]	BR_INSTR_RET	If 1, supports architectural index 5	Logical Processor
EAX[6]	BR_MISPRED_RET	If 1, supports architectural index 6	Logical Processor
EAX[7]	SLOTS	If 1, supports architectural index 7	Logical Processor
EAX[8]	BACKEND	If 1, supports architectural index 8	Logical Processor
EAX[9]	BADSPEC	If 1, supports architectural index 9	Logical Processor

EAX[10]	FRONTEND	If 1, supports architectural index 10	Logical Processor
EAX[11]	RETIRING	If 1, supports architectural index 11	Logical Processor
EAX[12]	LBR_INSERTS	If 1, supports architectural index 12	Logical Processor
EAX[31:13]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H.04H -- PEBS Capabilities

Table 21-76. Leaf 23H.04H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[2:0]	Reserved	Reserved.	
EBX[3]	ALLOW_IN_RECORD	If 1, indicates that the ALLOW_IN_RECORD bit is available in the IA32_PMC_GpN_CFG_C and IA32_PMC_Fxm_CFG_C MSRs.	Logical Processor
EBX[4]	CNTR_GP	If 1, indicates that counters group sub-group general-purpose counters is available.	Logical Processor
EBX[5]	CNTR_FIXED	If 1, indicates that counters group sub-group fixed-function counters is available.	Logical Processor
EBX[6]	CNTR_METRICS	If 1, indicates that counters group sub-group performance metrics is available.	Logical Processor
EBX[7]	Reserved	Reserved.	
EBX[9:8]	LBR	LBR group and both bits [41:40] are available.	Logical Processor
EBX[15:10]	Reserved	Reserved.	
EBX[23:16]	XER	XER group bits [50:49] and bits [55:53] are available. See Section 11.4.4, "XSAVEEnabled Registers Group," for XER fields.	Logical Processor
EBX[28:24]	Reserved	Reserved.	
EBX[29]	GPR	If 1, the GPR group is available.	Logical Processor
EBX[30]	AUX	If 1, the AUX group is available.	Logical Processor
EBX[31]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H.05H -- Arch PEBS GP and Fixed Counters supported

Table 21-77. Leaf 23H.05H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	GP_PEBS	Bit vector of general-purpose counters for which the Architectural PEBS mechanism is available (bit n == GP counter #n). If EAX[n] == 1, then the IA32_PMC_GPn_CFG_C MSR is available, and PEBS is supported on that counter; the PEBS_EN[63] field can be set; and the RELOAD[31:0] field can be set. Note that CPUID.23H.04H:EBX governs which adaptive group bits can be set.	Logical Processor
EBX[31:0]	GP_PDIST	General-purpose counters for which PEBS supports PDIST.	Logical Processor
ECX[31:0]	FIXED_PEBS	Bit vector of fixed-function counters for which the Architectural PEBS mechanism is available. If ECX[x] == 1, then the IA32_PMC_FXm_CFG_C MSR is available, and PEBS is supported; the PEBS_EN[63] field can be set; and the RELOAD[31:0] field can be set. Note that CPUID.23H.04H:EBX governs which adaptive group bits can be set.	Logical Processor
EDX[31:0]	FIXED_PDIST	Fixed-function counters for which PEBS supports PDIST.	Logical Processor

CPUID.24H -- Converged Vector ISA

When CPUID.24H, the processor returns Intel AVX10 converged vector ISA information. This leaf is supported when CPUID.07H.01H:EDX.AVX10[19] = 1.

- This leaf is valid if CPUID.07H.01H:EDX.AVX10[19] = 1 and MAX_LEAF ≥ 24H.
- The maximum sub-leaf value for ECX is specified in CPUID.24H.00H.EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.24H.00H -- Converged Vector ISA Main Sub-Leaf

Table 21-78. Leaf 24H.00H Converged Vector ISA

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum number of sub-leaves that are supported in leaf 24H.	Platform
EBX[7:0]	VECTOR_ISA_VERSION	Reports the Intel® AVX10 Converged Vector ISA version.	Platform
EBX[15:8]	Reserved	Reserved.	
EBX[18:16]	Reserved at 111	Always 111b. Earlier versions of this specification documented these bits as enumerating support for different vector lengths. Processors enumerating Intel® AVX10 support all vector lengths.	Platform
EBX[31:19]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.27H -- Intel® Resource Director Technology (Intel® RDT) Asymmetric Monitoring

CPUID.27H returns information for the Intel Resource Director Technology Monitoring capabilities with asymmetric topology.

As described below, software uses the bit vector returned in EDX by sub-leaf 00H to determine the available resource types (ResID) that can be monitored. This information is necessary for software to program the IA32_PQR_ASSOC and IA32_QM_EVTSEL MSRs such that Quality-of-Service data can be read afterwards from the IA32_QM_CTR MSR.

- This leaf is valid if CPUID.07H.01H:ECX.RDT_M_ASYM[0] = 1 and MAX_LEAF ≥ 27H.
- If the leaf is valid, sub-leaf 00H is always valid. Sub-leaf n (n ≥ 1) is only valid when (CPUID.27H.00H:EDX[n] == 1).
- This leaf must be read on each logical processor to determine the support on each processor.

CPUID.27H.00H -- Intel® RDT Asymmetric Monitoring Main Sub-Leaf

CPUID.27H.00H returns information about Intel RDT Monitoring Asymmetric.

Table 21-79. Leaf 27H.00H Intel® Resource Director Technology (Intel® RDT) Asymmetric Monitoring

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	MAX_RMID	Maximum range (zero-based) of RMID within this physical processor of all types.	Logical Processor
ECX[31:0]	Reserved	Reserved.	
EDX[0]	Reserved	Reserved.	
EDX[1]	L3_MON	If 1, supports L3 Cache Intel RDT Monitoring. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX.	Logical Processor
EDX[31:2]	Reserved	Reserved.	

CPUID.27H.00H returns information about the bit-vector representation of QoS monitoring resource types that are supported in the processor and maximum range of RMID values the processor can use to monitor of any supported resource types. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS monitoring capability available for that type.

CPUID.27H.01H -- L3 Cache Intel® Resource Director Technology Asymmetric Monitoring

CPUID.27H.01H returns information about L3 Cache Intel RDT monitoring asymmetric.

Table 21-80. Leaf 27H.01H Intel® Resource Director Technology (Intel® RDT) Asymmetric Monitoring

Register	Field Name	Description	Domain
EAX[7:0]	CTR_WIDTH	The counter width is encoded as an offset from 24b. A value of zero in this field indicates that 24-bit counters are supported. A value of 8 in this field indicates that 32-bit counters are supported.	Logical Processor
EAX[8]	RDT_M_OVF	If 1, supports an overflow bit in the IA32_QM_CTR MSR (bit 61).	Logical Processor

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

EAX[9]	IO_RDT_CMT	If 1, indicates the presence of non-CPU agent supporting Intel RDT CMT.	Logical Processor
EAX[10]	IO_RDT_MBM	If 1, indicates the presence of non-CPU agent supporting Intel RDT MBM support.	Logical Processor
EAX[31:11]	Reserved	Reserved.	
EBX[31:0]	CONV_FACTOR	Factor used to convert from reported IA32_QM_CTR value to derived occupancy metric (bytes) and Memory Bandwidth Monitoring (MBM) metrics.	Logical Processor
ECX[31:0]	MAX_RMID_L3	Maximum range (zero-based) of RMID of this resource type.	Logical Processor
EDX[0]	CMT_L3_OCCUP	If 1, supports L3 occupancy monitoring.	Logical Processor
EDX[1]	MBM_L3_TOTAL	If 1, supports L3 total bandwidth monitoring.	Logical Processor
EDX[2]	MBM_L3_LOCAL	If 1, supports L3 local bandwidth monitoring.	Logical Processor
EDX[31:3]	Reserved	Reserved.	

CPUID.28H -- Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

CPUID.28H returns information for Intel Resource Director Technology Allocation with asymmetric topology. This leaf is valid when CPUID.07H.01H:ECX.RDT_A_SYM[1] = 1. As described below, software uses the bit vector returned in EBX by subleaf 00H to determine the available QoS Enforcement (allocation) resource types that are supported in the processor. This information is necessary for software to configure each class of services using capability bit masks in the QoS Mask registers, IA32_resourceType_Mask_n.

- This leaf is valid if CPUID.07H.01H:ECX.RDT_A_SYM[1] = 1 and MAX_LEAF ≥ 28H.
- If the leaf is valid, sub-leaf 00H is always valid. Sub-leaf n (n ≥ 1) is only valid when (CPUID.28H.00H:EBX[n] == 1).

CPUID.28H.00H -- Intel® RDT Asymmetric Allocation Main Sub-Leaf

CPUID.28H.00H returns information about Intel RDT Allocation Asymmetric.

Table 21-81. Leaf 28H.00H Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[0]	Reserved	Reserved.	
EBX[1]	CAT_L3	Supports L3 Cache Allocation Technology if 1.	Logical Processor
EBX[2]	CAT_L2	Supports L2 Cache Allocation Technology if 1.	Logical Processor
EBX[3]	MBA	Supports Memory Bandwidth Allocation if 1.	Logical Processor
EBX[31:4]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.28H.00H returns information about the bit-vector representation of QoS Enforcement resource types that are supported in the processor. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS enforcement capability available for that type.

CPUID.28H.01H -- Asymmetric L3 Cache Allocation Technology

CPUID.28H.ResID=1 returns information about Asymmetric L3 Cache Allocation Technology.

Table 21-82. Leaf 28H.01H Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

Register	Field Name	Description	Domain
EAX[4:0]	CAT_L3_BITMASK_LENGTH	Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result.	Logical Processor
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	CAT_L3_CONTENTION	Bit-granular map of isolation/contention of allocation units.	Logical Processor
ECX[0]	Reserved	If 1, supports L3 CAT for non-CPU agents.	

ECX[1]	CAT_L3_NONCPU	N/A	Logical Processor
ECX[2]	CAT_L3_CDP	If 1, supports L3 Code and Data Prioritization Technology.	Logical Processor
ECX[3]	CAT_L3_NONCONTIG	If 1, supports non-contiguous capacity bitmasks. The bits that are set in the various IA32_L3_MASK_n registers do not have to be contiguous.	Logical Processor
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CAT_L3_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Logical Processor
EDX[31:16]	Reserved	Reserved.	

CPUID.28H.02H -- Asymmetric L2 Cache Allocation Technology

CPUID.28H.ResID=2 returns information about Asymmetric L2 Cache Allocation Technology.

Table 21-83. Leaf 28H.02H Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

Register	Field Name	Description	Domain
EAX[4:0]	CAT_L2_BITMASK_LENGTH	Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result.	Logical Processor
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	CAT_L2_CONTENTION	Bit-granular map of isolation/contention of allocation units.	
ECX[1:0]	Reserved	Reserved.	
ECX[2]	CAT_L2_CDP	If 1, supports L2 Code and Data Prioritization Technology.	
ECX[3]	CAT_L2_NONCONTIG	If 1, supports non-contiguous capacity bitmasks. The bits that are set in the various IA32_L2_MASK_n registers do not have to be contiguous.	Logical Processor
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CAT_L2_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Logical Processor
EDX[31:16]	Reserved	Reserved.	

CPUID.28H.03H -- Asymmetric Memory Bandwidth Allocation

CPUID.28H.ResID=3 returns information about Asymmetric Memory Bandwidth Allocation.

Table 21-84. Leaf 28H.03H Intel® Resource Director Technology (Intel® RDT) Asymmetric Allocation

Register	Field Name	Description	Domain
EAX[11:0]	MBA_MAX	Reports the maximum MBA throttling value supported for the corresponding ResID. Add one to the return value to get the result.	Logical Processor
EAX[31:12]	Reserved	Reserved.	

EBX[31:0]	Reserved	Reserved.	
ECX[0]	PER_THREAD_MBA	Per-thread MBA controls are supported.	Logical Processor
ECX[1]	Reserved	Reserved.	
ECX[2]	MBA_LINEAR	If 1, the response of the delay values is linear.	Logical Processor
ECX[31:3]	Reserved	Reserved.	
EDX[15:0]	MBA_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Logical Processor
EDX[31:16]	Reserved	Reserved.	

CPUID.80000000H -- Maximum Input Value for Extended Function CPUID Information

CPUID.80000000H returns the highest value the processor recognizes for returning extended processor information. The value is returned in the EAX register and is processor specific.

- This leaf is supported starting with Pentium 4.
- Processors prior to Pentium 4 treat bit 31 as 0, and this leaf returns the values from CPUID.00H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-85. Leaf 80000000H Maximum Input Value for Extended Function CPUID Information

Register	Field Name	Description	Domain
EAX[31:0]	MAX_EXTENDED_LEAF	Maximum input value for Extended Function CPUID Information.	Platform
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.80000001H -- Extended Processor Signature and Feature Bits

CPUID.80000001H returns information about extended processor signature and features bits.

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000001H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-86. Leaf 80000001H Extended Processor Signature and Feature Bits

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[0]	LAHF_SAHF_64	If 1, supports the LAHF/SAHF instructions in 64-bit mode. LAHF and SAHF are always available in other modes, regardless of the enumeration of this feature flag.	Platform
ECX[4:1]	Reserved	Reserved.	
ECX[5]	LZCNT	If 1, supports the LZCNT instruction.	Platform
ECX[7:6]	Reserved	Reserved.	
ECX[8]	PREFETCHW	If 1, supports the PREFETCHW instruction.	Platform
ECX[31:9]	Reserved	Reserved.	
EDX[10:0]	Reserved	Reserved.	
EDX[11]	SYSCALL_SYSRET_64	If 1, supports SYSCALL/SYSRET. Intel processors support SYSCALL and SYSRET only in 64-bit mode. This feature flag is always enumerated as 0 outside 64-bit mode.	Platform
EDX[19:12]	Reserved	Reserved.	
EDX[20]	EXECUTE_DIS	If 1, supports Execute Disable Bit.	Platform
EDX[25:21]	Reserved	Reserved.	
EDX[26]	PAGE_1GB	If 1, supports 1-GByte pages.	Platform
EDX[27]	RDTSCP	If 1, supports RDTSCP and IA32_TSC_AUX.	Platform
EDX[28]	Reserved	Reserved.	
EDX[29]	INTEL64	If 1, supports Intel® 64 Architecture.	Platform
EDX[31:30]	Reserved	Reserved.	

CPUID.80000002H -- Processor Brand String (Bytes 0 to 15)

CPUID.80000002H returns information about the Processor Brand String. For additional details on Processor Brand String, see Section 21.2, “Methods for Returning Branding Information Using CPUID.”

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000002H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-87. Leaf 80000002H Processor Brand String (Bytes 0 to 15)

Register	Field Name	Description	Domain
EAX[31:0]	BRAND_NAME_0	Processor brand string.	Platform
EBX[31:0]	BRAND_NAME_1	Processor brand string continued.	Platform
ECX[31:0]	BRAND_NAME_2	Processor brand string continued.	Platform
EDX[31:0]	BRAND_NAME_3	Processor brand string continued.	Platform

CPUID.80000003H -- Processor brand string (Bytes 16 to 31)

CPUID.80000003H returns information about the Processor Brand String. For additional details on Processor Brand String, see Section 21.2, “Methods for Returning Branding Information Using CPUID.”

- This leaf is valid if MAX_EXTENDED_LEAF $\geq$ 80000003H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-88. Leaf 80000003H Processor brand string (Bytes 16 to 31)

Register	Field Name	Description	Domain
EAX[31:0]	BRAND_NAME_4	Processor brand string continued.	Platform
EBX[31:0]	BRAND_NAME_5	Processor brand string continued.	Platform
ECX[31:0]	BRAND_NAME_6	Processor brand string continued.	Platform
EDX[31:0]	BRAND_NAME_7	Processor brand string continued.	Platform

CPUID.80000004H -- Processor brand string (Bytes 32 to 47)

CPUID.80000004H returns information about the Processor Brand String. For additional details on Processor Brand String, see Section 21.2, “Methods for Returning Branding Information Using CPUID.”

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000004H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-89. Leaf 80000004H Processor brand string (Bytes 32 to 47)

Register	Field Name	Description	Domain
EAX[31:0]	BRAND_NAME_8	Processor brand string continued.	Platform
EBX[31:0]	BRAND_NAME_9	Processor brand string continued.	Platform
ECX[31:0]	BRAND_NAME_10	Processor brand string continued.	Platform
EDX[31:0]	BRAND_NAME_11	Processor brand string continued.	Platform

CPUID.80000005H -- Reserved

This leaf is reserved and returns all zeroes.

Table 21-90. Leaf 80000005H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.80000006H -- Extended Function CPUID Information

CPUID.80000006H returns Extended Function CPUID information. The preferred method to enumerate caching information description>is to use CPUID.04H—Deterministic Cache Parameters.

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000006H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX

Table 21-91. Leaf 80000006H Extended Function CPUID Information

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[7:0]	L2_LINE_SIZE	Cache line size in bytes.	Logical Processor
ECX[11:8]	Reserved	Reserved.	
ECX[15:12]	L2_ASSOC	L2 associativity field. The L2 associativity field encodings are listed in the table below.	Logical Processor
ECX[31:16]	L2_SIZE	Cache size in 1K units.	Logical Processor
EDX[7:0]	Reserved	Reserved.	
EDX[31:8]	Reserved	Reserved.	

Table 21-92. L2 Associativity Field Encodings

Encoding Value	Description	Encoding Value	Description
00H	Disabled	08H	16 Ways
01H	1 Way (direct mapped)	09H	Reserved
02H	2 Ways	0AH	32 Ways
03H	Reserved	0BH	48 Ways
04H	4 Ways	0CH	64 Ways
05H	Reserved	0DH	96 Ways
06H	8 Ways	0EH	128 Ways
07H	See CPUID leaf 4 sub-leaf 2 ¹	0FH	Fully Associative

CPUID.80000007H -- Extended Function CPUID Information 1

CPUID.80000007H returns Extended Function CPUID information.

- This leaf is valid if MAX_EXTENDED_LEAF $\geq$ 80000007H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-93. Leaf 80000007H Extended Function CPUID Information 1

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[7:0]	Reserved	Reserved.	
EDX[8]	TSC_INVARIANT	If 1, supports Invariant TSC.	Platform
EDX[31:9]	Reserved	Reserved.	

CPUID.80000008H -- Extended Function CPUID Information 2

- CPUID.80000008H returns Extended Function CPUID information.
- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000008H.
 - This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-94. Leaf 80000008H Extended Function CPUID Information 2

Register	Field Name	Description	Domain
EAX[7:0]	PHYS_ADDR_SIZE	Number of physical-address bits. If TME-MK is enabled, the number of bits that can be used to address memory may be reduced by IA32_TME_ACTIVATE[35:32] .	Platform
EAX[15:8]	LIN_ADDR_SIZE	Number of linear-address bits.	Platform
EAX[23:16]	GUEST_PHYS_ADDR_SIZE	Number of guest-physical-address bits (for software operating in a virtual machine). If this field is zero, PHYS_ADDR_SIZE should be used. Intel processors return zero for this field. Software emulating CPUID may return a different value.	Platform
EAX[31:24]	Reserved	Reserved.	
EBX[8:0]	Reserved	Reserved.	
EBX[9]	WBNOINVD	If 1, supports the WBNOINVD instruction.	Platform
EBX[31:10]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

6. Updates to Chapter 3, Volume 2A

Change bars and text show changes to Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-L*.

Changes to this chapter:

- The following instructions were updated for Flexible Return Event Delivery (FRED):
 - CALL
 - CLRSSBSY
 - INT
 - IRET
 - JMP
- New instructions were added for FRED:
 - ERETS
 - ERETU
 - LKGS
- The Description for instruction CVTTSD2SI was corrected to state that when a conversion is inexact a truncated result is returned.
- The encodings for instruction INCSSPD and INCSSPQ was corrected in the opcode table to F3 0F AE /5 and F3 REX.W 0F AE /5, respectively.
- Instruction LDDQU VEX.256 encoded version, VLDDQU, was updated in the Operation section to zero the upper bits of the ZMM destination.

CHAPTER 3 INSTRUCTION SET REFERENCE, A-L

This chapter describes the instruction set for the Intel 64 and IA-32 architectures (A-L) in IA-32e, protected, virtual-8086, and real-address modes of operation. The set includes general-purpose, x87 FPU, MMX, SSE/SSE2/SSE3/SSSE3/SSE4, AESNI/PCLMULQDQ, AVX, and system instructions. See also Chapter 4, “Instruction Set Reference, M-U,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B; Chapter 5, “Instruction Set Reference, V,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2C; and Chapter 6, “Instruction Set Reference, W-Z,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

For each instruction, each operand combination is described. A description of the instruction and its operand, an operational description, a description of the effect of the instructions on flags in the EFLAGS register, and a summary of exceptions that can be generated are also provided.

3.1 INTERPRETING THE INSTRUCTION REFERENCE PAGES

This section describes the format of information contained in the instruction reference pages in this chapter. It explains notational conventions and abbreviations used in these sections.

3.1.1 Instruction Format

The following is an example of the format used for each instruction description in this chapter. The heading below introduces the example. The table below provides an example summary table.

CMC—Complement Carry Flag [this is an example]

Opcode	Instruction	Op/En	64/32-bit Mode	CPUID Feature Flag	Description
F5	CMC	Z0	V/V	N/A	Complement carry flag.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

3.1.1.1 Opcode Column in the Instruction Summary Table (Instructions without VEX Prefix)

The “Opcode” column in the table above shows the object code produced for each form of the instruction. When possible, codes are given as hexadecimal bytes in the same order in which they appear in memory. Definitions of entries other than hexadecimal bytes are as follows:

- **NP** — Indicates the use of 66/F2/F3 prefixes (beyond those already part of the instructions opcode) are not allowed with the instruction. Such use will either cause an invalid-opcode exception (#UD) or result in the encoding for a different instruction.
- **NF_x** — Indicates the use of F2/F3 prefixes (beyond those already part of the instructions opcode) are not allowed with the instruction. Such use will either cause an invalid-opcode exception (#UD) or result in the encoding for a different instruction.
- **REX.W** — Indicates the use of a REX prefix that affects operand size or instruction semantics. The ordering of the REX prefix and other optional/mandatory instruction prefixes are discussed Chapter 2. Note that REX prefixes that promote legacy instructions to 64-bit behavior are not listed explicitly in the opcode column.
- **/digit** — A digit between 0 and 7 indicates that the ModR/M byte of the instruction uses only the r/m (register or memory) operand. The reg field contains the digit that provides an extension to the instruction's opcode.
- **/r** — Indicates that the ModR/M byte of the instruction contains a register operand and an r/m operand.
- **cb, cw, cd, cp, co, ct** — A 1-byte (cb), 2-byte (cw), 4-byte (cd), 6-byte (cp), 8-byte (co) or 10-byte (ct) value following the opcode. This value is used to specify a code offset and possibly a new value for the code segment register.
- **ib, iw, id, io** — A 1-byte (ib), 2-byte (iw), 4-byte (id) or 8-byte (io) immediate operand to the instruction that follows the opcode, ModR/M bytes or scale-indexing bytes. The opcode determines if the operand is a signed value. All words, doublewords, and quadwords are given with the low-order byte first.
- **+rb, +rw, +rd, +ro** — Indicated the lower 3 bits of the opcode byte is used to encode the register operand without a modR/M byte. The instruction lists the corresponding hexadecimal value of the opcode byte with low 3 bits as 000b. In non-64-bit mode, a register code, from 0 through 7, is added to the hexadecimal value of the opcode byte. In 64-bit mode, indicates the four bit field of REX.b and opcode[2:0] field encodes the register operand of the instruction. “+ro” is applicable only in 64-bit mode. See Table 3-1 for the codes.
- **+i** — A number used in floating-point instructions when one of the operands is ST(i) from the FPU register stack. The number i (which can range from 0 to 7) is added to the hexadecimal byte given at the left of the plus sign to form a single opcode byte.

Table 3-1. Register Codes Associated With +rb, +rw, +rd, +ro

byte register			word register			dword register			quadword register (64-Bit Mode only)		
Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field
AL	None	0	AX	None	0	EAX	None	0	RAX	None	0
CL	None	1	CX	None	1	ECX	None	1	RCX	None	1
DL	None	2	DX	None	2	EDX	None	2	RDX	None	2
BL	None	3	BX	None	3	EBX	None	3	RBX	None	3
AH	Not encodable (N.E.)	4	SP	None	4	ESP	None	4	N/A	N/A	N/A
CH	N.E.	5	BP	None	5	EBP	None	5	N/A	N/A	N/A
DH	N.E.	6	SI	None	6	ESI	None	6	N/A	N/A	N/A
BH	N.E.	7	DI	None	7	EDI	None	7	N/A	N/A	N/A
SPL	Yes	4	SP	None	4	ESP	None	4	RSP	None	4
BPL	Yes	5	BP	None	5	EBP	None	5	RBP	None	5

Table 3-1. Register Codes Associated With +rb, +rw, +rd, +ro (Contd.)

byte register			word register			dword register			quadword register (64-Bit Mode only)		
Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field
SIL	Yes	6	SI	None	6	ESI	None	6	RSI	None	6
DIL	Yes	7	DI	None	7	EDI	None	7	RDI	None	7
Registers R8 - R15 (see below): Available in 64-Bit Mode Only											
R8B	Yes	0	R8W	Yes	0	R8D	Yes	0	R8	Yes	0
R9B	Yes	1	R9W	Yes	1	R9D	Yes	1	R9	Yes	1
R10B	Yes	2	R10W	Yes	2	R10D	Yes	2	R10	Yes	2
R11B	Yes	3	R11W	Yes	3	R11D	Yes	3	R11	Yes	3
R12B	Yes	4	R12W	Yes	4	R12D	Yes	4	R12	Yes	4
R13B	Yes	5	R13W	Yes	5	R13D	Yes	5	R13	Yes	5
R14B	Yes	6	R14W	Yes	6	R14D	Yes	6	R14	Yes	6
R15B	Yes	7	R15W	Yes	7	R15D	Yes	7	R15	Yes	7

3.1.1.2 Opcode Column in the Instruction Summary Table (Instructions with VEX prefix)

In the Instruction Summary Table, the Opcode column presents each instruction encoded using the VEX prefix in following form (including the modR/M byte if applicable, the immediate byte if applicable):

VEX.[128,256].[66,F2,F3].0F/0F3A/0F38.[W0,W1] opcode [/r] [/ib,/is4]

- **VEX** — Indicates the presence of the VEX prefix is required. The VEX prefix can be encoded using the three-byte form (the first byte is C4H), or using the two-byte form (the first byte is C5H). The two-byte form of VEX only applies to those instructions that do not require the following fields to be encoded: VEX.mmmmm, VEX.W, VEX.X, VEX.B. Refer to Section 2.3 for more detail on the VEX prefix.

The encoding of various sub-fields of the VEX prefix is described using the following notations:

- **128,256:** VEX.L field can be 0 (denoted by VEX.128, VEX.L0, or VEX.LZ) or 1 (denoted by VEX.256 or VEX.L1). The VEX.L field can be encoded using either the 2-byte or 3-byte form of the VEX prefix. The presence of the notation VEX.256 or VEX.128 in the opcode column should be interpreted as follows:
 - If VEX.256 is present in the opcode column: The semantics of the instruction must be encoded with VEX.L = 1. An attempt to encode this instruction with VEX.L = 0 can result in one of two situations: (a) if VEX.128 version is defined, the processor will behave according to the defined VEX.128 behavior; (b) an #UD occurs if there is no VEX.128 version defined.
 - If VEX.128 is present in the opcode column but there is no VEX.256 version defined for the same opcode byte: Two situations apply: (a) For VEX-encoded, 128-bit SIMD integer instructions, software must encode the instruction with VEX.L = 0. The processor will treat the opcode byte encoded with VEX.L = 1 by causing an #UD exception; (b) For VEX-encoded, 128-bit packed floating-point instructions, software must encode the instruction with VEX.L = 0. The processor will treat the opcode byte encoded with VEX.L = 1 by causing an #UD exception (e.g., VMOVLPS).
 - If VEX.L0 or VEX.L1 is present in the opcode column: The specified VEX.L value is required for encoding this instruction but does not have the connotation of specifying vector length.
 - If VEX.LIG is present in the opcode column: The VEX.L value is ignored. This generally applies to VEX-encoded scalar SIMD floating-point instructions. Scalar SIMD floating-point instruction can be distinguished from the mnemonic of the instruction. Generally, the last two letters of the instruction mnemonic would be either "SS", "SD", or "SI" for SIMD floating-point conversion instructions.
 - If VEX.LZ is present in the opcode column: The VEX.L must be encoded to be 0B, an #UD occurs if VEX.L is not zero.

- **66,F2,F3:** The presence or absence of these values map to the VEX.pp field encodings. If absent, this corresponds to VEX.pp=00B. If present, the corresponding VEX.pp value affects the “opcode” byte in the same way as if a SIMD prefix (66H, F2H or F3H) does to the ensuing opcode byte. Thus a non-zero encoding of VEX.pp may be considered as an implied 66H/F2H/F3H prefix. The VEX.pp field may be encoded using either the 2-byte or 3-byte form of the VEX prefix.
- **0F,0F3A,0F38:** The presence maps to a valid encoding of the VEX.mmmmm field. Only three encoded values of VEX.mmmmm are defined as valid, corresponding to the escape byte sequence of 0FH, 0F3AH, and 0F38H. The effect of a valid VEX.mmmmm encoding on the ensuing opcode byte is same as if the corresponding escape byte sequence on the ensuing opcode byte for non-VEX encoded instructions. Thus a valid encoding of VEX.mmmmm may be considered as an implied escape byte sequence of either 0FH, 0F3AH or 0F38H. The VEX.mmmmm field must be encoded using the 3-byte form of VEX prefix.
- **0F,0F3A,0F38 and 2-byte/3-byte VEX:** The presence of 0F3A and 0F38 in the opcode column implies that opcode can only be encoded by the three-byte form of VEX. The presence of 0F in the opcode column does not preclude the opcode to be encoded by the two-byte of VEX if the semantics of the opcode does not require any subfield of VEX not present in the two-byte form of the VEX prefix.
- **W0:** VEX.W=0.
- **W1:** VEX.W=1.
- The presence of W0/W1 in the opcode column applies to two situations: (a) it is treated as an extended opcode bit, (b) the instruction semantics support an operand size promotion to 64-bit of a general-purpose register operand or a 32-bit memory operand. The presence of W1 in the opcode column implies the opcode must be encoded using the 3-byte form of the VEX prefix. The presence of W0 in the opcode column does not preclude the opcode to be encoded using the C5H form of the VEX prefix, if the semantics of the opcode does not require other VEX subfields not present in the two-byte form of the VEX prefix. Please see Section 2.3 on the subfield definitions within VEX.
- **WIG:** can use C5H form (if not requiring VEX.mmmmm) or VEX.W value is ignored in the C4H form of VEX prefix.
- If WIG is present, the instruction may be encoded using either the two-byte form or the three-byte form of VEX. When encoding the instruction using the three-byte form of VEX, the value of VEX.W is ignored.
- **opcode** — Instruction opcode.
- **/is4** — An 8-bit immediate byte is present containing a source register specifier in either imm8[7:4] (for 64-bit mode) or imm8[6:4] (for 32-bit mode), and instruction-specific payload in imm8[3:0].
- In general, the encoding of VEX.R, VEX.X, VEX.B field are not shown explicitly in the opcode column. The encoding scheme of VEX.R, VEX.X, VEX.B fields must follow the rules defined in Section 2.3.

EVEX.[128,256,512,LLIG].[66,F2,F3].0F/0F3A/0F38.[W0,W1,WIG] opcode [/r] [/ib]

- **EVEX** — The EVEX prefix is encoded using the four-byte form (the first byte is 62H). Refer to Section 2.7.1 for more detail on the EVEX prefix.

The encoding of various sub-fields of the EVEX prefix is described using the following notations:

- **128, 256, 512, LLIG:** This corresponds to the vector length; three values are allowed by EVEX: 512-bit, 256-bit and 128-bit. Alternatively, vector length is ignored (LIG) for certain instructions; this typically applies to scalar instructions operating on one data element of a vector register.
- **66,F2,F3:** The presence of these value maps to the EVEX.pp field encodings. The corresponding VEX.pp value affects the “opcode” byte in the same way as if a SIMD prefix (66H, F2H or F3H) does to the ensuing opcode byte. Thus a non-zero encoding of VEX.pp may be considered as an implied 66H/F2H/F3H prefix.
- **0F,0F3A,0F38:** The presence maps to a valid encoding of the EVEX.mmm field. Only three encoded values of EVEX.mmm are defined as valid, corresponding to the escape byte sequence of 0FH, 0F3AH, and 0F38H. The effect of a valid EVEX.mmm encoding on the ensuing opcode byte is the same as if the corresponding escape byte sequence on the ensuing opcode byte for non-EVEX encoded instructions. Thus a valid encoding of EVEX.mmm may be considered as an implied escape byte sequence of either 0FH, 0F3AH or 0F38H.
- **W0:** EVEX.W=0.

- **W1:** EVEX.W=1.
- **WIG:** EVEX.W bit ignored
- **opcode** — Instruction opcode.
- In general, the encoding of EVEX.R and R', EVEX.X and X', and EVEX.B and B' fields are not shown explicitly in the opcode column.

NOTE

Previously, the terms NDS, NDD, and DDS were used in instructions with an EVEX (or VEX) prefix. These terms indicated that the vvvv field was valid for encoding, and specified register usage. These terms are no longer necessary and are redundant with the instruction operand encoding tables provided with each instruction. The instruction operand encoding tables give explicit details on all operands, indicating where every operand is stored and if they are read or written. If vvvv is not listed as an operand in the instruction operand encoding table, then EVEX (or VEX) vvvv must be 0b1111.

3.1.1.3 Instruction Column in the Opcode Summary Table

The “Instruction” column gives the syntax of the instruction statement as it would appear in an ASM386 program. The following is a list of the symbols used to represent operands in the instruction statements:

- **rel8** — A relative address in the range from 128 bytes before the end of the instruction to 127 bytes after the end of the instruction.
- **rel16, rel32** — A relative address within the same code segment as the instruction assembled. The rel16 symbol applies to instructions with an operand-size attribute of 16 bits; the rel32 symbol applies to instructions with an operand-size attribute of 32 bits.
- **ptr16:16, ptr16:32** — A far pointer, typically to a code segment different from that of the instruction. The notation 16:16 indicates that the value of the pointer has two parts. The value to the left of the colon is a 16-bit selector or value destined for the code segment register. The value to the right corresponds to the offset within the destination segment. The ptr16:16 symbol is used when the instruction's operand-size attribute is 16 bits; the ptr16:32 symbol is used when the operand-size attribute is 32 bits.
- **r8** — One of the byte general-purpose registers: AL, CL, DL, BL, AH, CH, DH, BH, BPL, SPL, DIL, and SIL; or one of the byte registers (R8B - R15B) available when using REX.R and 64-bit mode.
- **r16** — One of the word general-purpose registers: AX, CX, DX, BX, SP, BP, SI, DI; or one of the word registers (R8-R15) available when using REX.R and 64-bit mode.
- **r32** — One of the doubleword general-purpose registers: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI; or one of the doubleword registers (R8D - R15D) available when using REX.R in 64-bit mode.
- **r64** — One of the quadword general-purpose registers: RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8-R15. These are available when using REX.R and 64-bit mode.
- **imm8** — An immediate byte value. The imm8 symbol can be a signed number between -128 and +127 inclusive; an unsigned number between 0 and 255 inclusive; or a bitmap when an instruction uses its individual bits. For instructions in which imm8 is combined with a word or doubleword operand, the immediate value is sign-extended to form a word or doubleword. The upper byte of the word is filled with the topmost bit of the immediate value.
- **imm16** — An immediate word value used for instructions whose operand-size attribute is 16 bits. This is a number between -32,768 and +32,767 inclusive.
- **imm32** — An immediate doubleword value used for instructions whose operand-size attribute is 32 bits. It allows the use of a number between +2,147,483,647 and -2,147,483,648 inclusive.
- **imm64** — An immediate quadword value used for instructions whose operand-size attribute is 64 bits. The value allows the use of a number between +9,223,372,036,854,775,807 and -9,223,372,036,854,775,808 inclusive.
- **/ib** — A single-byte value.

- **r/m8** — A byte operand that is either the contents of a byte general-purpose register (AL, CL, DL, BL, AH, CH, DH, BH, BPL, SPL, DIL, and SIL) or a byte from memory. Byte registers R8B - R15B are available using REX.R in 64-bit mode.
- **r/m16** — A word general-purpose register or memory operand used for instructions whose operand-size attribute is 16 bits. The word general-purpose registers are: AX, CX, DX, BX, SP, BP, SI, DI. The contents of memory are found at the address provided by the effective address computation. Word registers R8W - R15W are available using REX.R in 64-bit mode.
- **r/m32** — A doubleword general-purpose register or memory operand used for instructions whose operand-size attribute is 32 bits. The doubleword general-purpose registers are: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI. The contents of memory are found at the address provided by the effective address computation. Doubleword registers R8D - R15D are available when using REX.R in 64-bit mode.
- **r/m64** — A quadword general-purpose register or memory operand used for instructions whose operand-size attribute is 64 bits when using REX.W. Quadword general-purpose registers are: RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8-R15; these are available only in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **reg** — A general-purpose register used for instructions when the width of the register does not matter to the semantics of the operation of the instruction. The register can be r16, r32, or r64.
- **m** — A 16-, 32- or 64-bit operand in memory.
- **m8** — A byte operand in memory, usually expressed as a variable or array name, but pointed to by the DS:(E)SI or ES:(E)DI registers. In 64-bit mode, it is pointed to by the RSI or RDI registers.
- **m16** — A word operand in memory, usually expressed as a variable or array name, but pointed to by the DS:(E)SI or ES:(E)DI registers. This nomenclature is used only with the string instructions.
- **m32** — A doubleword operand in memory. The contents of memory are found at the address provided by the effective address computation.
- **m64** — A memory quadword operand in memory.
- **m128** — A memory double quadword operand in memory.
- **m16:16, m16:32 & m16:64** — A memory operand containing a far pointer composed of two numbers. The number to the left of the colon corresponds to the pointer's segment selector. The number to the right corresponds to its offset.
- **m16&32, m16&16, m32&32, m16&64** — A memory operand consisting of data item pairs whose sizes are indicated on the left and the right side of the ampersand. All memory addressing modes are allowed. The m16&16 and m32&32 operands are used by the BOUND instruction to provide an operand containing an upper and lower bounds for array indices. The m16&32 operand is used by LIDT and LGDT to provide a word with which to load the limit field, and a doubleword with which to load the base field of the corresponding GDTR and IDTR registers. The m16&64 operand is used by LIDT and LGDT in 64-bit mode to provide a word with which to load the limit field, and a quadword with which to load the base field of the corresponding GDTR and IDTR registers.
- **m80bcd** — A Binary Coded Decimal (BCD) operand in memory, 80 bits.
- **mooffs8, mooffs16, mooffs32, mooffs64** — A simple memory variable (memory offset) of type byte, word, or doubleword used by some variants of the MOV instruction. The actual address is given by a simple offset relative to the segment base. No ModR/M byte is used in the instruction. The number shown with mooffs indicates its size, which is determined by the address-size attribute of the instruction.
- **Sreg** — A segment register. The segment register bit assignments are ES = 0, CS = 1, SS = 2, DS = 3, FS = 4, and GS = 5.
- **m32fp, m64fp, m80fp** — A single precision, double precision, and double extended-precision (respectively) floating-point operand in memory. These symbols designate floating-point values that are used as operands for x87 FPU floating-point instructions.
- **m16int, m32int, m64int** — A word, doubleword, and quadword integer (respectively) operand in memory. These symbols designate integers that are used as operands for x87 FPU integer instructions.
- **ST or ST(0)** — The top element of the FPU register stack.
- **ST(i)** — The i^{th} element from the top of the FPU register stack ($i := 0$ through 7).
- **mm** — An MMX register. The 64-bit MMX registers are: MM0 through MM7.

- **mm/m32** — The low order 32 bits of an MMX register or a 32-bit memory operand. The 64-bit MMX registers are: MM0 through MM7. The contents of memory are found at the address provided by the effective address computation.
- **mm/m64** — An MMX register or a 64-bit memory operand. The 64-bit MMX registers are: MM0 through MM7. The contents of memory are found at the address provided by the effective address computation.
- **xmm** — An XMM register. The 128-bit XMM registers are: XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode.
- **xmm/m32** — An XMM register or a 32-bit memory operand. The 128-bit XMM registers are XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **xmm/m64** — An XMM register or a 64-bit memory operand. The 128-bit SIMD floating-point registers are XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **xmm/m128** — An XMM register or a 128-bit memory operand. The 128-bit XMM registers are XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **<XMM0>** — Indicates implied use of the XMM0 register.
When there is ambiguity, xmm1 indicates the first source operand using an XMM register and xmm2 the second source operand using an XMM register.
Some instructions use the XMM0 register as the third source operand, indicated by <XMM0>. The use of the third XMM register operand is implicit in the instruction encoding and does not affect the ModR/M encoding.
- **ymm** — A YMM register. The 256-bit YMM registers are: YMM0 through YMM7; YMM8 through YMM15 are available in 64-bit mode.
- **m256** — A 32-byte operand in memory. This nomenclature is used only with AVX instructions.
- **ymm/m256** — A YMM register or 256-bit memory operand.
- **<YMM0>** — Indicates use of the YMM0 register as an implicit argument.
- **bnd** — A 128-bit bounds register. BND0 through BND3.
- **mib** — A memory operand using SIB addressing form, where the index register is not used in address calculation, Scale is ignored. Only the base and displacement are used in effective address calculation.
- **m512** — A 64-byte operand in memory.
- **zmm/m512** — A ZMM register or 512-bit memory operand.
- **{k1}{z}** — A mask register used as instruction writemask. The 64-bit k registers are: k1 through k7. Writemask specification is available exclusively via EVEX prefix. The masking can either be done as a merging-masking, where the old values are preserved for masked out elements or as a zeroing masking. The type of masking is determined by using the EVEX.z bit.
- **{k1}** — Without {z}: a mask register used as instruction writemask for instructions that do not allow zeroing-masking but support merging-masking. This corresponds to instructions that require the value of the aaa field to be different than 0 (e.g., gather) and store-type instructions which allow only merging-masking.
- **k1** — A mask register used as a regular operand (either destination or source). The 64-bit k registers are: k0 through k7.
- **mV** — A vector memory operand; the operand size is dependent on the instruction.
- **vm32{x,y,z}** — A vector array of memory operands specified using VSIB memory addressing. The array of memory addresses are specified using a common base register, a constant scale factor, and a vector index register with individual elements of 32-bit index value in an XMM register (vm32x), a YMM register (vm32y) or a ZMM register (vm32z).
- **vm64{x,y,z}** — A vector array of memory operands specified using VSIB memory addressing. The array of memory addresses are specified using a common base register, a constant scale factor, and a vector index register with individual elements of 64-bit index value in an XMM register (vm64x), a YMM register (vm64y) or a ZMM register (vm64z).

- **zmm/m512/m32bcst** — An operand that can be a ZMM register, a 512-bit memory location or a 512-bit vector loaded from a 32-bit memory location.
- **zmm/m512/m64bcst** — An operand that can be a ZMM register, a 512-bit memory location or a 512-bit vector loaded from a 64-bit memory location.
- **<ZMM0>** — Indicates use of the ZMM0 register as an implicit argument.
- **{er}** — Indicates support for embedded rounding control, which is only applicable to the register-register form of the instruction. This also implies support for SAE (Suppress All Exceptions).
- **{sae}** — Indicates support for SAE (Suppress All Exceptions). This is used for instructions that support SAE, but do not support embedded rounding control.
- **SRC1** — Denotes the first source operand in the instruction syntax of an instruction encoded with the VEX/EVEX prefix and having two or more source operands.
- **SRC2** — Denotes the second source operand in the instruction syntax of an instruction encoded with the VEX/EVEX prefix and having two or more source operands.
- **SRC3** — Denotes the third source operand in the instruction syntax of an instruction encoded with the VEX/EVEX prefix and having three source operands.
- **SRC** — The source in a single-source instruction.
- **DST** — The destination in an instruction. This field is encoded by reg_field.

In the instruction encoding, the MODRM byte is represented several ways depending on the role it plays. The MODRM byte has 3 fields: 2-bit MODRM.MOD field, a 3-bit MODRM.REG field and a 3-bit MODRM.RM field. When all bits of the MODRM byte have fixed values for an instruction, the 2-hex nibble value of that byte is presented after the opcode in the encoding boxes on the instruction description pages. When only some fields of the MODRM byte must contain fixed values, those values are specified as follows:

- If only the MODRM.MOD must be 0b11, and MODRM.REG and MODRM.RM fields are unrestricted, this is denoted as **11:rrr:bbb**. The **rrr** correspond to the 3-bits of the MODRM.REG field and the **bbb** correspond to the 3-bits of the MODRM.RM field.
- If the MODRM.MOD field is constrained to be a value other than 0b11, i.e., it must be one of 0b00, 0b01, or 0b10, then we use the notation **!(11)**.
- If the MODRM.REG field had a specific required value, e.g., 0b101, that would be denoted as **mm:101:bbb**.

3.1.1.4 Operand Encoding Column in the Instruction Summary Table

The “operand encoding” column is abbreviated as Op/En in the Instruction Summary table heading. Instruction operand encoding information is provided for each assembly instruction syntax using a letter to cross reference to a row entry in the operand encoding definition table that follows the instruction summary table. The operand encoding table in each instruction reference page lists each instruction operand (according to each instruction syntax and operand ordering shown in the instruction column) relative to the ModRM byte, VEX.vvvv field or additional operand encoding placement.

EVEX encoded instructions employ compressed $\text{disp8} \times N$ encoding of the displacement bytes, where N is defined in Table 2-36 and Table 2-37, according to tuple types. The tuple type for an instruction is listed in the operand encoding definition table where applicable.

NOTES

- The letters in the Op/En column of an instruction apply ONLY to the encoding definition table immediately following the instruction summary table.
- In the encoding definition table, the letter ‘r’ within a pair of parenthesis denotes the content of the operand will be read by the processor. The letter ‘w’ within a pair of parenthesis denotes the content of the operand will be updated by the processor.

3.1.1.5 64/32-bit Mode Column in the Instruction Summary Table

The “64/32-bit Mode” column indicates whether the opcode sequence is supported in (a) 64-bit mode or (b) the Compatibility mode and other IA-32 modes that apply in conjunction with the CPUID feature flag associated specific instruction extensions.

The 64-bit mode support is to the left of the 'slash' and has the following notation:

- **V** — Supported.
- **I** — Not supported.
- **N.E.** — Indicates an instruction syntax is not encodable in 64-bit mode (it may represent part of a sequence of valid instructions in other modes).
- **N.P.** — Indicates the REX prefix does not affect the legacy instruction in 64-bit mode.
- **N.I.** — Indicates the opcode is treated as a new instruction in 64-bit mode.
- **N.S.** — Indicates an instruction syntax that requires an address override prefix in 64-bit mode and is not supported. Using an address override prefix in 64-bit mode may result in model-specific execution behavior.

The Compatibility/Legacy Mode support is to the right of the 'slash' and has the following notation:

- **V** — Supported.
- **I** — Not supported.
- **N.E.** — Indicates an Intel 64 instruction mnemonics/syntax that is not encodable; the opcode sequence is not applicable as an individual instruction in compatibility mode or IA-32 mode. The opcode may represent a valid sequence of legacy IA-32 instructions.

3.1.1.6 CPUID Support Column in the Instruction Summary Table

The fourth column holds abbreviated CPUID feature flags (e.g., appropriate bit in CPUID.01H:ECX, CPUID.01H:EDX for SSE/SSE2/SSE3/SSSE3/SSE4.1/SSE4.2/AESNI/PCLMULQDQ/AVX/RDRAND support) that indicate processor support for the instruction. If the corresponding flag is '0', the instruction will #UD.

3.1.1.7 Description Column in the Instruction Summary Table

The "Description" column briefly explains forms of the instruction.

3.1.1.8 Description Section

Each instruction is then described by number of information sections. The "Description" section describes the purpose of the instructions and required operands in more detail.

Summary of terms that may be used in the description section:

- **Legacy SSE** — Refers to SSE, SSE2, SSE3, SSSE3, SSE4, AESNI, PCLMULQDQ, and any future instruction sets referencing XMM registers and encoded without a VEX prefix.
- **VEX.vvvv** — The VEX bit field specifying a source or destination register (in 1's complement form).
- **rm_field** — shorthand for the ModR/M *r/m* field and any REX.B.
- **reg_field** — shorthand for the ModR/M *reg* field and any REX.R.

3.1.1.9 Operation Section

The "Operation" section contains an algorithm description (frequently written in pseudo-code) for the instruction. Algorithms are composed of the following elements:

- Comments are enclosed within the symbol pairs "(" and ")".
- Compound statements are enclosed in keywords, such as: IF, THEN, ELSE, and FI for an if statement; DO and OD for a do statement; or CASE... OF for a case statement.
- A register name implies the contents of the register. A register name enclosed in brackets implies the contents of the location whose address is contained in that register. For example, ES:[DI] indicates the contents of the location whose ES segment relative address is in register DI. [SI] indicates the contents of the address contained in register SI relative to the SI register's default segment (DS) or the overridden segment.
- Parentheses around the "E" in a general-purpose register name, such as (E)SI, indicates that the offset is read from the SI register if the address-size attribute is 16, from the ESI register if the address-size attribute is 32.

Parentheses around the “R” in a general-purpose register name, (R)SI, in the presence of a 64-bit register definition such as (R)SI, indicates that the offset is read from the 64-bit RSI register if the address-size attribute is 64.

- Brackets are used for memory operands where they mean that the contents of the memory location is a segment-relative offset. For example, [SRC] indicates that the content of the source operand is a segment-relative offset.
- A := B indicates that the value of B is assigned to A.
- The symbols =, ≠, >, <, ≥, and ≤ are relational operators used to compare two values: meaning equal, not equal, greater or equal, less or equal, respectively. A relational expression such as A = B is TRUE if the value of A is equal to B; otherwise it is FALSE.
- The expression “<< COUNT” and “>> COUNT” indicates that the destination operand should be shifted left or right by the number of bits indicated by the count operand.

The following identifiers are used in the algorithmic descriptions:

- **OperandSize and AddressSize** — The OperandSize identifier represents the operand-size attribute of the instruction, which is 16, 32 or 64-bits. The AddressSize identifier represents the address-size attribute, which is 16, 32 or 64-bits. For example, the following pseudo-code indicates that the operand-size attribute depends on the form of the MOV instruction used.

```
IF Instruction = MOVW
    THEN OperandSize := 16;
ELSE
    IF Instruction = MOVD
        THEN OperandSize := 32;
    ELSE
        IF Instruction = MOVQ
            THEN OperandSize := 64;
        FI;
    FI;
FI;
```

See “Operand-Size and Address-Size Attributes” in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for guidelines on how these attributes are determined.

- **StackAddrSize** — Represents the stack address-size attribute associated with the instruction, which has a value of 16, 32 or 64-bits. See “Address-Size Attribute for Stack” in Chapter 6, “Procedure Calls, Interrupts, and Exceptions,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.
- **SRC** — Represents the source operand.
- **DEST** — Represents the destination operand.
- **MAXVL** — The maximum vector register width pertaining to the instruction. This is not the vector-length encoding in the instruction’s encoding but is instead determined by the current value of XCR0. For details, refer to the table below. Note that the value of MAXVL is the largest of the features enabled. Future processors may define new bits in XCR0 whose setting may imply other values for MAXVL.

MAXVL Definition

XCR0 Component	MAXVL
XCR0.SSE	128
XCR0.AVX	256
XCR0.{ZMM_Hi256, Hi16_ZMM, OPMASK}	512

The following functions are used in the algorithmic descriptions:

- **ZeroExtend(value)** — Returns a value zero-extended to the operand-size attribute of the instruction. For example, if the operand-size attribute is 32, zero extending a byte value of –10 converts the byte from F6H to

a doubleword value of 000000F6H. If the value passed to the ZeroExtend function and the operand-size attribute are the same size, ZeroExtend returns the value unaltered.

- **SignExtend(value)** — Returns a value sign-extended to the operand-size attribute of the instruction. For example, if the operand-size attribute is 32, sign extending a byte containing the value –10 converts the byte from F6H to a doubleword value of FFFFFFF6H. If the value passed to the SignExtend function and the operand-size attribute are the same size, SignExtend returns the value unaltered.
- **SaturateSignedWordToSignedByte** — Converts a signed 16-bit value to a signed 8-bit value. If the signed 16-bit value is less than –128, it is represented by the saturated value –128 (80H); if it is greater than 127, it is represented by the saturated value 127 (7FH).
- **SaturateSignedDwordToSignedWord** — Converts a signed 32-bit value to a signed 16-bit value. If the signed 32-bit value is less than –32768, it is represented by the saturated value –32768 (8000H); if it is greater than 32767, it is represented by the saturated value 32767 (7FFFH).
- **SaturateSignedWordToUnsignedByte** — Converts a signed 16-bit value to an unsigned 8-bit value. If the signed 16-bit value is less than zero, it is represented by the saturated value zero (00H); if it is greater than 255, it is represented by the saturated value 255 (FFH).
- **SaturateToSignedByte** — Represents the result of an operation as a signed 8-bit value. If the result is less than –128, it is represented by the saturated value –128 (80H); if it is greater than 127, it is represented by the saturated value 127 (7FH).
- **SaturateToSignedWord** — Represents the result of an operation as a signed 16-bit value. If the result is less than –32768, it is represented by the saturated value –32768 (8000H); if it is greater than 32767, it is represented by the saturated value 32767 (7FFFH).
- **SaturateToUnsignedByte** — Represents the result of an operation as a signed 8-bit value. If the result is less than zero it is represented by the saturated value zero (00H); if it is greater than 255, it is represented by the saturated value 255 (FFH).
- **SaturateToUnsignedWord** — Represents the result of an operation as a signed 16-bit value. If the result is less than zero it is represented by the saturated value zero (00H); if it is greater than 65535, it is represented by the saturated value 65535 (FFFFH).
- **LowOrderWord(DEST * SRC)** — Multiplies a word operand by a word operand and stores the least significant word of the doubleword result in the destination operand.
- **HighOrderWord(DEST * SRC)** — Multiplies a word operand by a word operand and stores the most significant word of the doubleword result in the destination operand.
- **Push(value)** — Pushes a value onto the stack. The number of bytes pushed is determined by the operand-size attribute of the instruction. See the “Operation” subsection of the “PUSH—Push Word, Doubleword, or Quadword Onto the Stack” section in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B.
- **Pop()** — removes the value from the top of the stack and returns it. The statement `EAX := Pop();` assigns to EAX the 32-bit value from the top of the stack. Pop will return either a word, a doubleword or a quadword depending on the operand-size attribute. See the “Operation” subsection in the “POP—Pop a Value From the Stack” section of Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B.
- **PopRegisterStack** — Marks the FPU ST(0) register as empty and increments the FPU register stack pointer (TOP) by 1.
- **Switch-Tasks** — Performs a task switch.
- **Bit(BitBase, BitOffset)** — Returns the value of a bit within a bit string. The bit string is a sequence of bits in memory or a register. Bits are numbered from low-order to high-order within registers and within memory bytes. If the BitBase is a register, the BitOffset can be in the range 0 to [15, 31, 63] depending on the mode and register size. See Figure 3-1: the function `Bit[RAX, 21]` is illustrated.

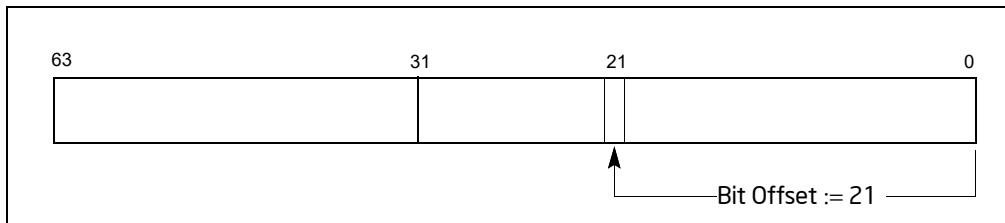


Figure 3-1. Bit Offset for BIT[RAX, 21]

If BitBase is a memory address, the BitOffset has different ranges depending on the operand size (see Table 3-2).

Table 3-2. Range of Bit Positions Specified by Bit Offset Operands

Operand Size	Immediate BitOffset	Register BitOffset
16	0 to 15	-2^{15} to $2^{15} - 1$
32	0 to 31	-2^{31} to $2^{31} - 1$
64	0 to 63	-2^{63} to $2^{63} - 1$

The addressed bit is numbered (Offset MOD 8) within the byte at address (BitBase + (BitOffset DIV 8)) where DIV is signed division with rounding towards negative infinity and MOD returns a positive number (see Figure 3-2).

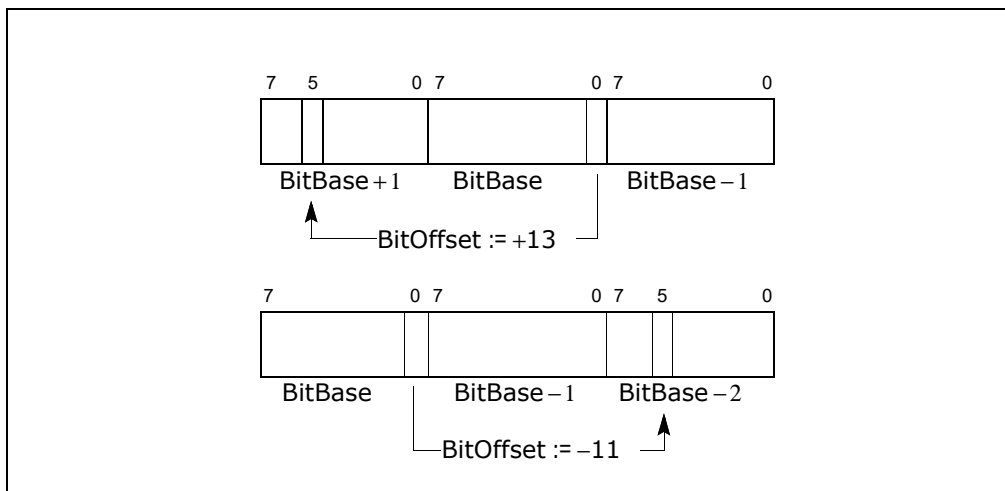


Figure 3-2. Memory Bit Indexing

3.1.1.10 Intel® C/C++ Compiler Intrinsics Equivalents Section

The Intel C/C++ compiler intrinsic functions give access to the full power of the Intel Architecture Instruction Set, while allowing the compiler to optimize register allocation and instruction scheduling for faster execution. Most of these functions are associated with a single IA instruction, although some may generate multiple instructions or different instructions depending upon how they are used. In particular, these functions are used to invoke instructions that perform operations on vector registers that can hold multiple data elements. These SIMD instructions use the following data types.

- `__m128`, `__m256`, and `__m512` can represent 4, 8, or 16 packed single precision floating-point values, and are used with the vector registers and SSE, AVX, or AVX-512 instruction set extension families. The `__m128` data type is also used with various single precision floating-point scalar instructions that perform calculations using

only the lowest 32 bits of a vector register; the remaining bits of the result come from one of the sources or are set to zero depending upon the instruction.

- `__m128d`, `__m256d`, and `__m512d` can represent 2, 4, or 8 packed double precision floating-point values, and are used with the vector registers and SSE, AVX, or AVX-512 instruction set extension families. The `__m128d` data type is also used with various double precision floating-point scalar instructions that perform calculations using only the lowest 64 bits of a vector register; the remaining bits of the result come from one of the sources or are set to zero depending upon the instruction.
- `__m128i`, `__m256i`, and `__m512i` can represent integer data in bytes, words, doublewords, quadwords, and occasionally larger data types.

Each of these data types incorporates in its name the number of bits it can hold. For example, the `__m128` type holds 128 bits, and because each single precision floating-point value is 32 bits long the `__m128` type holds (128/32) or four values. Normally the compiler will allocate memory for these data types on an even multiple of the size of the type. Such aligned memory locations may be faster to read and write than locations at other addresses.

These SIMD data types are not basic Standard C data types or C++ objects, so they may be used only with the assignment operator, passed as function arguments, and returned from a function call. If you access the internal members of these types directly, or indirectly by using them in a union, there may be side effects affecting optimization, so it is recommended to use them only with the SIMD instruction intrinsic functions described in this manual or the Intel C/C++ compiler documentation.

Many intrinsic functions names are prefixed with an indicator of the vector length and suffixed by an indicator of the vector element data type, although some functions do not follow the rules below. The prefixes are:

- `_mm_` indicates that the function operates on 128-bit (or sometimes 64-bit) vectors.
- `_mm256_` indicates the function operates on 256-bit vectors.
- `_mm512_` indicates that the function operates on 512-bit vectors.

The suffixes include:

- `_ps`, which indicates a function that operates on packed single precision floating-point data. Packed single precision floating-point data corresponds to arrays of the C/C++ type *float* with either 4, 8 or 16 elements. Values of this type can be loaded from an array using the `_mm_loadu_ps`, `_mm256_loadu_ps`, or `_mm512_loadu_ps` functions, or created from individual values using `_mm_set_ps`, `_mm256_set_ps`, or `_mm512_set_ps` functions, and they can be stored in an array using `_mm_storeu_ps`, `_mm256_storeu_ps`, or `_mm512_storeu_ps`.
- `_ss`, which indicates a function that operates on scalar single precision floating-point data. Single precision floating-point data corresponds to the C/C++ type *float*, and values of type *float* can be converted to type `__m128` for use with these functions using the `_mm_set_ss` function, and converted back using the `_mm_cvtss_f32` function. When used with functions that operate on packed single precision floating-point data the scalar element corresponds with the first packed value.
- `_pd`, which indicates a function that operates on packed double precision floating-point data. Packed double precision floating-point data corresponds to arrays of the C/C++ type *double* with either 2, 4, or 8 elements. Values of this type can be loaded from an array using the `_mm_loadu_pd`, `_mm256_loadu_pd`, or `_mm512_loadu_pd` functions, or created from individual values using `_mm_set_pd`, `_mm256_set_pd`, or `_mm512_set_pd` functions, and they can be stored in an array using `_mm_storeu_pd`, `_mm256_storeu_pd`, or `_mm512_storeu_pd`.
- `_sd`, which indicates a function that operates on scalar double precision floating-point data. Double precision floating-point data corresponds to the C/C++ type *double*, and values of type *double* can be converted to type `__m128d` for use with these functions using the `_mm_set_sd` function, and converted back using the `_mm_cvtsd_f64` function. When used with functions that operate on packed double precision floating-point data the scalar element corresponds with the first packed value.
- `_epi8`, which indicates a function that operates on packed 8-bit signed integer values. Packed 8-bit signed integers correspond to an array of *signed char* with 16, 32 or 64 elements. Values of this type can be created from individual elements using `_mm_set_epi8`, `_mm256_set_epi8`, or `_mm512_set_epi8` functions.
- `_epi16`, which indicates a function that operates on packed 16-bit signed integer values. Packed 16-bit signed integers correspond to an array of *short* with 8, 16 or 32 elements. Values of this type can be created from individual elements using `_mm_set_epi16`, `_mm256_set_epi16`, or `_mm512_set_epi16` functions.

- `_epi32`, which indicates a function that operates on packed 32-bit signed integer values. Packed 32-bit signed integers correspond to an array of *int* with 4, 8 or 16 elements. Values of this type can be created from individual elements using `_mm_set_epi32`, `_mm256_set_epi32`, or `_mm512_set_epi32` functions.
- `_epi64`, which indicates a function that operates on packed 64-bit signed integer values. Packed 64-bit signed integers correspond to an array of *long long* (or *long* if it is a 64-bit data type) with 2, 4 or 8 elements. Values of this type can be created from individual elements using `_mm_set_epi32`, `_mm256_set_epi32`, or `_mm512_set_epi32` functions.
- `_epu8`, which indicates a function that operates on packed 8-bit unsigned integer values. Packed 8-bit unsigned integers correspond to an array of *unsigned char* with 16, 32 or 64 elements.
- `_epu16`, which indicates a function that operates on packed 16-bit unsigned integer values. Packed 16-bit unsigned integers correspond to an array of *unsigned short* with 8, 16 or 32 elements.
- `_epu32`, which indicates a function that operates on packed 32-bit unsigned integer values. Packed 32-bit unsigned integers correspond to an array of *unsigned* with 4, 8 or 16 elements.
- `_epu64`, which indicates a function that operates on packed 64-bit unsigned integer values. Packed 64-bit unsigned integers correspond to an array of *unsigned long long* (or *unsigned long* if it is a 64-bit data type) with 2, 4 or 8 elements.
- `_si128`, which indicates a function that operates on a single 128-bit value of type `__m128i`.
- `_si256`, which indicates a function that operates on a single a 256-bit value of type `__m256i`.
- `_si512`, which indicates a function that operates on a single a 512-bit value of type `__m512i`.

Values of any packed integer type can be loaded from an array using the `_mm_loadu_si128`, `_mm256_loadu_si256`, or `_mm512_loadu_si512` functions, and they can be stored in an array using `_mm_storeu_si128`, `_mm256_storeu_si256`, or `_mm512_storeu_si512`.

These functions and data types are used with the SSE, AVX, and AVX-512 instruction set extension families. In addition there are similar functions that correspond to MMX instructions. These are less frequently used because they require additional state management, and only operate on 64-bit packed integer values.

The declarations of Intel C/C++ compiler intrinsic functions may reference some non-standard data types, such as `__int64`. The C Standard header `stdint.h` defines similar platform-independent types, and the documentation for that header gives characteristics that apply to corresponding non-standard types according to the following table.

Table 3-3. Standard and Non-Standard Data Types

Non-standard Type	Standard Type (from <code>stdint.h</code>)
<code>__int64</code>	<code>int64_t</code>
<code>unsigned __int64</code>	<code>uint64_t</code>
<code>__int32</code>	<code>int32_t</code>
<code>unsigned __int32</code>	<code>uint32_t</code>
<code>__int16</code>	<code>int16_t</code>
<code>unsigned __int16</code>	<code>uint16_t</code>

For a more detailed description of each intrinsic function and additional information related to its usage, refer to the online Intel Intrinsics Guide, <https://software.intel.com/sites/landingpage/IntrinsicsGuide>.

3.1.1.11 Flags Affected Section

The “Flags Affected” section lists the flags in the EFLAGS register that are affected by the instruction. When a flag is cleared, it is equal to 0; when it is set, it is equal to 1. The arithmetic and logical instructions usually assign values to the status flags in a uniform manner (see Appendix A, “EFLAGS Cross-Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). Non-conventional assignments are described in the “Operation” section. The values of flags listed as **undefined** may be changed by the instruction in an indeterminate manner. Flags that are not listed are unchanged by the instruction.

3.1.1.12 FPU Flags Affected Section

The floating-point instructions have an “FPU Flags Affected” section that describes how each instruction can affect the four condition code flags of the FPU status word.

3.1.1.13 Protected Mode Exceptions Section

The “Protected Mode Exceptions” section lists the exceptions that can occur when the instruction is executed in protected mode and the reasons for the exceptions. Each exception is given a mnemonic that consists of a pound sign (#) followed by two letters and an optional error code in parentheses. For example, #GP(0) denotes a general protection exception with an error code of 0. Table 3-4 associates each two-letter mnemonic with the corresponding exception vector and name. See Chapter 6, “Procedure Calls, Interrupts, and Exceptions,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, for a detailed description of the exceptions.

Application programmers should consult the documentation provided with their operating systems to determine the actions taken when exceptions occur.

Table 3-4. Intel 64® and IA-32 General Exceptions

Vector	Name	Source	Protected Mode ¹	Real Address Mode	Virtual 8086 Mode
0	#DE—Divide Error	DIV and IDIV instructions.	Yes	Yes	Yes
1	#DB—Debug	Any code or data reference.	Yes	Yes	Yes
3	#BP—Breakpoint	INT3 instruction.	Yes	Yes	Yes
4	#OF—Overflow	INTO instruction.	Yes	Yes	Yes
5	#BR—BOUND Range Exceeded	BOUND instruction.	Yes	Yes	Yes
6	#UD—Invalid Opcode (Undefined Opcode)	UD instruction or reserved opcode.	Yes	Yes	Yes
7	#NM—Device Not Available (No Math Coprocessor)	Floating-point or WAIT/FWAIT instruction.	Yes	Yes	Yes
8	#DF—Double Fault	Any instruction that can generate an exception, an NMI, or an INTR.	Yes	Yes	Yes
10	#TS—Invalid TSS	Task switch or TSS access.	Yes	No	Yes
11	#NP—Segment Not Present	Loading segment registers or accessing system segments.	Yes	No	Yes
12	#SS—Stack Segment Fault	Stack operations and SS register loads.	Yes	Yes	Yes
13	#GP—General Protection ²	Any memory reference and other protection checks.	Yes	Yes	Yes
14	#PF—Page Fault	Any memory reference.	Yes	No	Yes
16	#MF—Floating-Point Error (Math Fault)	Floating-point or WAIT/FWAIT instruction.	Yes	Yes	Yes
17	#AC—Alignment Check	Any data reference in memory.	Yes	No	Yes
18	#MC—Machine Check	Model dependent machine check errors.	Yes	Yes	Yes
19	#XM—SIMD Floating-Point Numeric Error	SSE/SSE2/SSE3 floating-point instructions.	Yes	Yes	Yes
20	#VE—Virtualization Exception	EPT violations ³	Yes	No	No

Table 3-4. Intel 64® and IA-32 General Exceptions (Contd.)

Vector	Name	Source	Protected Mode ¹	Real Address Mode	Virtual 8086 Mode
21	#CP—Control Protection Exception	RET, IRET, RSTORSSP, and SETSSBSY instructions can generate this exception. When CET indirect branch tracking is enabled, this exception can be generated due to a missing ENDBRANCH instruction at target of an indirect call or jump.	Yes	No	No

NOTES:

1. Apply to protected mode, compatibility mode, and 64-bit mode.
2. In the real-address mode, vector 13 is the segment overrun exception.
3. This exception can occur only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.

3.1.1.14 Real-Address Mode Exceptions Section

The “Real-Address Mode Exceptions” section lists the exceptions that can occur when the instruction is executed in real-address mode (see Table 3-4).

3.1.1.15 Virtual-8086 Mode Exceptions Section

The “Virtual-8086 Mode Exceptions” section lists the exceptions that can occur when the instruction is executed in virtual-8086 mode (see Table 3-4).

3.1.1.16 Floating-Point Exceptions Section

The “Floating-Point Exceptions” section lists exceptions that can occur when an x87 FPU floating-point instruction is executed. All of these exception conditions result in a floating-point error exception (#MF, exception 16) being generated. Table 3-5 associates a one- or two-letter mnemonic with the corresponding exception name. See “Floating-Point Exception Conditions” in Chapter 8 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for a detailed description of these exceptions.

Table 3-5. x87 FPU Floating-Point Exceptions

Mnemonic	Name	Source
#IS #IA	Floating-point invalid operation: - Stack overflow or underflow - Invalid arithmetic operation	- x87 FPU stack overflow or underflow - Invalid FPU arithmetic operation
#Z	Floating-point divide-by-zero	Divide-by-zero
#D	Floating-point denormal operand	Source operand that is a denormal number
#O	Floating-point numeric overflow	Overflow in result
#U	Floating-point numeric underflow	Underflow in result
#P	Floating-point inexact result (precision)	Inexact result (precision)

3.1.1.17 SIMD Floating-Point Exceptions Section

The “SIMD Floating-Point Exceptions” section lists exceptions that can occur when an SSE/SSE2/SSE3 floating-point instruction is executed. All of these exception conditions result in a SIMD floating-point error exception (#XM, exception 19) being generated. Table 3-6 associates a one-letter mnemonic with the corresponding exception name. For a detailed description of these exceptions, refer to “SSE and SSE2 Exceptions”, in Chapter 11 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

Table 3-6. SIMD Floating-Point Exceptions

Mnemonic	Name	Source
#I	Floating-point invalid operation	Invalid arithmetic operation or source operand
#Z	Floating-point divide-by-zero	Divide-by-zero
#D	Floating-point denormal operand	Source operand that is a denormal number
#O	Floating-point numeric overflow	Overflow in result
#U	Floating-point numeric underflow	Underflow in result
#P	Floating-point inexact result	Inexact result (precision)

3.1.1.18 Compatibility Mode Exceptions Section

This section lists exceptions that occur within compatibility mode.

3.1.1.19 64-Bit Mode Exceptions Section

This section lists exceptions that occur within 64-bit mode.

3.2 INTEL® AMX CONSIDERATIONS

The following implementation parameters and helper functions are applicable to the Intel® AMX instructions.

3.2.1 Implementation Parameters

The parameters are reported via CPUID.1DH. Index 0 reports all zeros for all fields.

```
define palette_table[id]:
    uint16_t total_tile_bytes
    uint16_t bytes_per_tile
    uint16_t bytes_per_row
    uint16_t max_names
    uint16_t max_rows
```

The tile parameters are set by LDTILECFG or XRSTOR* of TILECFG:

```
define tile[tid]:
    byte rows
    word colsb // bytes_per_row
    bool valid
```

3.2.2 Helper Functions

The helper functions used in Intel AMX instructions are defined below.


```

define write_row_and_zero(treg, r, data, nbytes):
    for j in 0 ...nbytes-1:
        treg.row[r].byte[j] := data.byte[j]

    // zero the rest of the row
    for j in nbytes ... palette_table[tilecfg.palette_id].bytes_per_row-1:
        treg.row[r].byte[j] := 0

define zero_upper_rows(treg, r):
    for i in r ... palette_table[tilecfg.palette_id].max_rows-1:
        for j in 0 ... palette_table[tilecfg.palette_id].bytes_per_row-1:
            treg.row[i].byte[j] := 0

define zero_tilecfg_start():
    tilecfg.start_row :=0

define zero_all_tile_data():
    if XCR0[TILEDATA]:
        b := CPUID.0DH.TILEDATA:EAX // size of feature
        for j in 0 ... b:
            TILEDATA.byte[j] := 0

define xcr0_supports_palette(palette_id):
    if palette_id == 0:
        return 1
    elif palette_id == 1:
        if XCR0[TILECFG] and XCR0[TILEDATA]:
            return 1
    return 0

```

3.3 INSTRUCTIONS (A-L)

The remainder of this chapter provides descriptions of Intel 64 and IA-32 instructions (A-L). See also: Chapter 4, “Instruction Set Reference, M-U,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B; Chapter 5, “Instruction Set Reference, V,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2C; and Chapter 6, “Instruction Set Reference, W-Z,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

CALL—Call Procedure

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
E8 cw	CALL rel16	D	N.S.	Valid	Call near, relative, displacement relative to next instruction.
E8 cd	CALL rel32	D	Valid	Valid	Call near, relative, displacement relative to next instruction. 32-bit displacement sign extended to 64-bits in 64-bit mode.
FF /2	CALL r/m16	M	N.E.	Valid	Call near, absolute indirect, address given in r/m16.
FF /2	CALL r/m32	M	N.E.	Valid	Call near, absolute indirect, address given in r/m32.
FF /2	CALL r/m64	M	Valid	N.E.	Call near, absolute indirect, address given in r/m64.
9A cd	CALL ptr16:16	D	Invalid	Valid	Call far, absolute, address given in operand.
9A cp	CALL ptr16:32	D	Invalid	Valid	Call far, absolute, address given in operand.
FF /3	CALL m16:16	M	Valid	Valid	Call far, absolute indirect address given in m16:16. In 32-bit mode: if selector points to a gate, then RIP = 32-bit zero extended displacement taken from gate; else RIP = zero extended 16-bit offset from far pointer referenced in the instruction.
FF /3	CALL m16:32	M	Valid	Valid	In 64-bit mode: If selector points to a gate, then RIP = 64-bit displacement taken from gate; else RIP = zero extended 32-bit offset from far pointer referenced in the instruction.
REX.W FF /3	CALL m16:64	M	Valid	N.E.	In 64-bit mode: If selector points to a gate, then RIP = 64-bit displacement taken from gate; else RIP = 64-bit offset from far pointer referenced in the instruction.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
D	Offset	N/A	N/A	N/A
M	ModRM:r/m (r)	N/A	N/A	N/A

Description

Saves procedure linking information on the stack and branches to the called procedure specified using the target operand. The target operand specifies the address of the first instruction in the called procedure. The operand can be an immediate value, a general-purpose register, or a memory location.

This instruction can be used to execute four types of calls:

- **Near Call** — A call to a procedure in the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intra-segment call.
- **Far Call** — A call to a procedure located in a different segment than the current code segment, sometimes referred to as an inter-segment call.
- **Inter-privilege-level far call** — A far call to a procedure in a segment at a different privilege level than that of the currently executing program or procedure.
- **Task switch** — A call to a procedure located in a different task.

The latter two call types (inter-privilege-level call and task switch) can only be executed in protected mode. See “Calling Procedures Using Call and RET” in Chapter 6 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for additional information on near, far, and inter-privilege-level calls. See Chapter 10, “Task Management,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, for information on performing task switches with the CALL instruction.

Near Call. When executing a near call, the processor pushes the value of the EIP register (which contains the offset of the instruction following the CALL instruction) on the stack (for use later as a return-instruction pointer). The processor then branches to the address in the current code segment specified by the target operand. The target operand specifies either an absolute offset in the code segment (an offset from the base of the code segment) or a relative offset (a signed displacement relative to the current value of the instruction pointer in the EIP register; this value points to the instruction following the CALL instruction). The CS register is not changed on near calls.

For a near call absolute, an absolute offset is specified indirectly in a general-purpose register or a memory location (*r/m16*, *r/m32*, or *r/m64*). The operand-size attribute determines the size of the target operand (16, 32 or 64 bits). When in 64-bit mode, the operand size for near call (and all near branches) is forced to 64-bits. Absolute offsets are loaded directly into the EIP(RIP) register. If the operand size attribute is 16, the upper two bytes of the EIP register are cleared, resulting in a maximum instruction pointer size of 16 bits. When accessing an absolute offset indirectly using the stack pointer [ESP] as the base register, the base value used is the value of the ESP before the instruction executes.

A relative offset (*rel16* or *rel32*) is generally specified as a label in assembly code. But at the machine code level, it is encoded as a signed, 16- or 32-bit immediate value. This value is added to the value in the EIP(RIP) register. In 64-bit mode the relative offset is always a 32-bit immediate value which is sign extended to 64-bits before it is added to the value in the RIP register for the target calculation. As with absolute offsets, the operand-size attribute determines the size of the target operand (16, 32, or 64 bits). In 64-bit mode the target operand will always be 64-bits because the operand size is forced to 64-bits for near branches.

Far Calls in Real-Address or Virtual-8086 Mode. When executing a far call in real- address or virtual-8086 mode, the processor pushes the current value of both the CS and EIP registers on the stack for use as a return-instruction pointer. The processor then performs a “far branch” to the code segment and offset specified with the target operand for the called procedure. The target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). With the pointer method, the segment and offset of the called procedure is encoded in the instruction using a 4-byte (16-bit operand size) or 6-byte (32-bit operand size) far address immediate. With the indirect method, the target operand specifies a memory location that contains a 4-byte (16-bit operand size) or 6-byte (32-bit operand size) far address. The operand-size attribute determines the size of the offset (16 or 32 bits) in the far address. The far address is loaded directly into the CS and EIP registers. If the operand-size attribute is 16, the upper two bytes of the EIP register are cleared.

Far Calls in Protected Mode. When the processor is operating in protected mode, the CALL instruction can be used to perform the following types of far calls:

- Far call to the same privilege level
- Far call to a different privilege level (inter-privilege level call)
- Task switch (far call to another task)

In protected mode, the processor always uses the segment selector part of the far address to access the corresponding descriptor in the GDT or LDT. The descriptor type (code segment, call gate, task gate, or TSS) and access rights determine the type of call operation to be performed.

If the selected descriptor is for a code segment, a far call to a code segment at the same privilege level is performed. (If the selected code segment is at a different privilege level and the code segment is non-conforming, a general-protection exception is generated.) A far call to the same privilege level in protected mode is very similar to one carried out in real-address or virtual-8086 mode. The target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The operand-size attribute determines the size of the offset (16 or 32 bits) in the far address. The new code segment selector and its descriptor are loaded into CS register; the offset from the instruction is loaded into the EIP register.

A call gate (described in the next paragraph) can also be used to perform a far call to a code segment at the same privilege level. Using this mechanism provides an extra level of indirection and is the preferred method of making calls between 16-bit and 32-bit code segments.

When executing an inter-privilege-level far call, the code segment for the procedure being called must be accessed through a call gate. The segment selector specified by the target operand identifies the call gate. The target operand can specify the call gate segment selector either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The processor obtains the segment selector for the new code segment and the new instruction pointer (offset) from the call gate descriptor. (The offset from the target operand is ignored when a call gate is used.)

On inter-privilege-level calls, the processor switches to the stack for the privilege level of the called procedure. The segment selector for the new stack segment is specified in the TSS for the currently running task. The branch to the new code segment occurs after the stack switch. (Note that when using a call gate to perform a far call to a segment at the same privilege level, no stack switch occurs.) On the new stack, the processor pushes the segment selector and stack pointer for the calling procedure's stack, an optional set of parameters from the calling procedure's stack, and the segment selector and instruction pointer for the calling procedure's code segment. (A value in the call gate descriptor determines how many parameters to copy to the new stack.) Finally, the processor branches to the address of the procedure being called within the new code segment.

Executing a task switch with the CALL instruction is similar to executing a call through a call gate. The target operand specifies the segment selector of the task gate for the new task activated by the switch (the offset in the target operand is ignored). The task gate in turn points to the TSS for the new task, which contains the segment selectors for the task's code and stack segments. Note that the TSS also contains the EIP value for the next instruction that was to be executed before the calling task was suspended. This instruction pointer value is loaded into the EIP register to re-start the calling task.

The CALL instruction can also specify the segment selector of the TSS directly, which eliminates the indirection of the task gate. See Chapter 10, "Task Management," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for information on the mechanics of a task switch.

When **execution of a CALL instruction effects a task switch**, the nested task flag (NT) is set in the EFLAGS register and the new TSS's previous task link field is loaded with the old task's TSS selector. Code is expected to suspend this nested task by executing an IRET instruction which, because the NT flag is set, automatically uses the previous task link to return to the calling task. (See "Task Linking" in Chapter 10 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for information on nested tasks.) Switching tasks with the CALL instruction differs in this regard from JMP instruction. JMP does not set the NT flag and therefore does not expect an IRET instruction to suspend the task.

Mixing 16-Bit and 32-Bit Calls. When making far calls between 16-bit and 32-bit code segments, use a call gate. If the far call is from a 32-bit code segment to a 16-bit code segment, the call should be made from the first 64 KBytes of the 32-bit code segment. This is because the operand-size attribute of the instruction is set to 16, so only a 16-bit return address offset can be saved. Also, the call should be made using a 16-bit call gate so that 16-bit values can be pushed on the stack. See Chapter 24, "Mixing 16-Bit and 32-Bit Code," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, for more information.

Far Calls in Compatibility Mode. When the processor is operating in compatibility mode, the CALL instruction can be used to perform the following types of far calls:

- Far call to the same privilege level, remaining in compatibility mode
- Far call to the same privilege level, transitioning to 64-bit mode
- Far call to a different privilege level (inter-privilege level call), transitioning to 64-bit mode

Note that a CALL instruction cannot be used to cause a task switch in compatibility mode since task switches are not supported in IA-32e mode.

In compatibility mode, the processor always uses the segment selector part of the far address to access the corresponding descriptor in the GDT or LDT. The descriptor type (code segment, call gate) and access rights determine the type of call operation to be performed.

If the selected descriptor is for a code segment, a far call to a code segment at the same privilege level is performed. (If the selected code segment is at a different privilege level and the code segment is non-conforming, a general-protection exception is generated.) A far call to the same privilege level in compatibility mode is very similar to one carried out in protected mode. The target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The operand-size attribute determines the size of the offset (16 or 32 bits) in the far address. The new code segment selector and its descriptor are loaded into CS register and the offset from the instruction is loaded into the EIP register. The difference is that 64-bit mode may be entered. This is specified by the L bit in the new code segment descriptor.

Note that a 64-bit call gate (described in the next paragraph) can also be used to perform a far call to a code segment at the same privilege level. However, using this mechanism requires that the target code segment descriptor have the L bit set, causing an entry to 64-bit mode.

When executing an inter-privilege-level far call, the code segment for the procedure being called must be accessed through a 64-bit call gate. The segment selector specified by the target operand identifies the call gate. The target

operand can specify the call gate segment selector either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The processor obtains the segment selector for the new code segment and the new instruction pointer (offset) from the 16-byte call gate descriptor. (The offset from the target operand is ignored when a call gate is used.)

On inter-privilege-level calls, the processor switches to the stack for the privilege level of the called procedure. The segment selector for the new stack segment is set to NULL. The new stack pointer is specified in the TSS for the currently running task. The branch to the new code segment occurs after the stack switch. (Note that when using a call gate to perform a far call to a segment at the same privilege level, an implicit stack switch occurs as a result of entering 64-bit mode. The SS selector is unchanged, but stack segment accesses use a segment base of 0x0, the limit is ignored, and the default stack size is 64-bits. The full value of RSP is used for the offset, of which the upper 32-bits are undefined.) On the new stack, the processor pushes the segment selector and stack pointer for the calling procedure's stack and the segment selector and instruction pointer for the calling procedure's code segment. (Parameter copy is not supported in IA-32e mode.) Finally, the processor branches to the address of the procedure being called within the new code segment.

Near/(Far) Calls in 64-bit Mode. When the processor is operating in 64-bit mode, the CALL instruction can be used to perform the following types of far calls:

- Far call to the same privilege level, transitioning to compatibility mode
- Far call to the same privilege level, remaining in 64-bit mode
- Far call to a different privilege level (inter-privilege level call), remaining in 64-bit mode

Note that in this mode the CALL instruction can not be used to cause a task switch in 64-bit mode since task switches are not supported in IA-32e mode.

In 64-bit mode, the processor always uses the segment selector part of the far address to access the corresponding descriptor in the GDT or LDT. The descriptor type (code segment, call gate) and access rights determine the type of call operation to be performed.

If the selected descriptor is for a code segment, a far call to a code segment at the same privilege level is performed. (If the selected code segment is at a different privilege level and the code segment is non-conforming, a general-protection exception is generated.) A far call to the same privilege level in 64-bit mode is very similar to one carried out in compatibility mode. The target operand specifies an absolute far address indirectly with a memory location (*m16:16*, *m16:32* or *m16:64*). The form of CALL with a direct specification of absolute far address is not defined in 64-bit mode. The operand-size attribute determines the size of the offset (16, 32, or 64 bits) in the far address. The new code segment selector and its descriptor are loaded into the CS register; the offset from the instruction is loaded into the EIP register. The new code segment may specify entry either into compatibility or 64-bit mode, based on the L bit value.

A 64-bit call gate (described in the next paragraph) can also be used to perform a far call to a code segment at the same privilege level. However, using this mechanism requires that the target code segment descriptor have the L bit set.

When executing an inter-privilege-level far call, the code segment for the procedure being called must be accessed through a 64-bit call gate. The segment selector specified by the target operand identifies the call gate. The target operand can only specify the call gate segment selector indirectly with a memory location (*m16:16*, *m16:32* or *m16:64*). The processor obtains the segment selector for the new code segment and the new instruction pointer (offset) from the 16-byte call gate descriptor. (The offset from the target operand is ignored when a call gate is used.)

On inter-privilege-level calls, the processor switches to the stack for the privilege level of the called procedure. The segment selector for the new stack segment is set to NULL. The new stack pointer is specified in the TSS for the currently running task. The branch to the new code segment occurs after the stack switch.

Note that when using a call gate to perform a far call to a segment at the same privilege level, an implicit stack switch occurs as a result of entering 64-bit mode. The SS selector is unchanged, but stack segment accesses use a segment base of 0x0, the limit is ignored, and the default stack size is 64-bits. (The full value of RSP is used for the offset.) On the new stack, the processor pushes the segment selector and stack pointer for the calling procedure's stack and the segment selector and instruction pointer for the calling procedure's code segment. (Parameter copy is not supported in IA-32e mode.) Finally, the processor branches to the address of the procedure being called within the new code segment.

Refer to Chapter 6, “Procedure Calls, Interrupts, and Exceptions,” and Chapter 18, “Control-flow Enforcement Technology (CET),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for CET details.

When FRED transitions are enabled, an execution of far CALL that references a call gate causes a general-protection exception, as does an execution of far CALL that would enter compatibility mode when CPL is 0.

Instruction ordering. Instructions following a far call may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the far call have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Instructions sequentially following a near indirect CALL instruction (i.e., those not at the target) may be executed speculatively. If software needs to prevent this (e.g., in order to prevent a speculative execution side channel), then an LFENCE instruction opcode can be placed after the near indirect CALL in order to block speculative execution.

Operation

IF near call

THEN IF near relative call

THEN

IF OperandSize = 64

THEN

tempDEST := SignExtend(DEST); (* DEST is rel32 *)

tempRIP := RIP + tempDEST;

IF stack not large enough for a 8-byte return address

THEN #SS(0); FI;

Push(RIP);

IF ShadowStackEnabled(CPL) AND DEST != 0

ShadowStackPush8B(RIP);

FI;

RIP := tempRIP;

FI;

IF OperandSize = 32

THEN

tempEIP := EIP + DEST; (* DEST is rel32 *)

IF tempEIP is not within code segment limit THEN #GP(0); FI;

IF stack not large enough for a 4-byte return address

THEN #SS(0); FI;

Push(EIP);

IF ShadowStackEnabled(CPL) AND DEST != 0

ShadowStackPush4B(EIP);

FI;

EIP := tempEIP;

FI;

IF OperandSize = 16

THEN

tempIP := (EIP + DEST) AND 0000FFFFH; (* DEST is rel16 *)

IF tempIP is not within code segment limit THEN #GP(0); FI;

IF stack not large enough for a 2-byte return address

THEN #SS(0); FI;

Push(IP);

IF ShadowStackEnabled(CPL) AND DEST != 0

(* IP is zero extended and pushed as a 32 bit value on shadow stack *)

ShadowStackPush4B(IP);

FI;

```

        EIP := tempEIP;
    FI;
ELSE (* Near absolute call *)
    IF OperandSize = 64
        THEN
            tempRIP := DEST; (* DEST is r/m64 *)
            IF stack not large enough for a 8-byte return address
                THEN #SS(0); FI;
            Push(RIP);
            IF ShadowStackEnabled(CPL)
                ShadowStackPush8B(RIP);
            FI;
            RIP := tempRIP;
        FI;
    IF OperandSize = 32
        THEN
            tempEIP := DEST; (* DEST is r/m32 *)
            IF tempEIP is not within code segment limit THEN #GP(0); FI;
            IF stack not large enough for a 4-byte return address
                THEN #SS(0); FI;
            Push(EIP);
            IF ShadowStackEnabled(CPL)
                ShadowStackPush4B(EIP);
            FI;
            EIP := tempEIP;
        FI;
    IF OperandSize = 16
        THEN
            tempEIP := DEST AND 0000FFFFH; (* DEST is r/m16 *)
            IF tempEIP is not within code segment limit THEN #GP(0); FI;
            IF stack not large enough for a 2-byte return address
                THEN #SS(0); FI;
            Push(IP);
            IF ShadowStackEnabled(CPL)
                (* IP is zero extended and pushed as a 32 bit value on shadow stack *)
                ShadowStackPush4B(IP);
            FI;
            EIP := tempEIP;
        FI;
    FI;rel/abs
IF (Call near indirect, absolute indirect)
    IF EndbranchEnabledAndNotSuppressed(CPL)
        IF CPL = 3
            THEN
                IF ( no 3EH prefix OR IA32_U_CET.NO_TRACK_EN == 0 )
                    THEN
                        IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
                    FI;
            ELSE
                IF ( no 3EH prefix OR IA32_S_CET.NO_TRACK_EN == 0 )
                    THEN
                        IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
                    FI;
            FI;
        FI;
    FI;

```

```

    FI;
  FI;
FI; near

IF far call and (PE = 0 or (PE = 1 and VM = 1)) (* Real-address or virtual-8086 mode *)
  THEN
    IF OperandSize = 32
      THEN
        IF stack not large enough for a 6-byte return address
          THEN #SS(0); FI;
        IF DEST[31:16] is not zero THEN #GP(0); FI;
        Push(CS); (* Padded with 16 high-order bits *)
        Push(EIP);
        CS := DEST[47:32]; (* DEST is ptr16:32 or [m16:32] *)
        EIP := DEST[31:0]; (* DEST is ptr16:32 or [m16:32] *)
      ELSE (* OperandSize = 16 *)
        IF stack not large enough for a 4-byte return address
          THEN #SS(0); FI;
        Push(CS);
        Push(IP);
        CS := DEST[31:16]; (* DEST is ptr16:16 or [m16:16] *)
        EIP := DEST[15:0]; (* DEST is ptr16:16 or [m16:16]; clear upper 16 bits *)
      FI;
    FI;
  FI;

```

```

IF far call and (PE = 1 and VM = 0) (* Protected mode or IA-32e Mode, not virtual-8086 mode*)
  THEN
    IF segment selector in target operand NULL
      THEN #GP(0); FI;
    IF segment selector index not within descriptor table limits
      THEN #GP(new code segment selector); FI;
    Read type and access rights of selected segment descriptor;
    IF IA32_EFER.LMA = 0
      THEN
        IF segment type is not a conforming or nonconforming code segment, call gate, task gate, or TSS
          THEN #GP(segment selector); FI;
        ELSE
          IF segment type is not a conforming or nonconforming code segment or 64-bit call gate
            THEN #GP(segment selector); FI;
        FI;
      Depending on type and access rights:
        GO TO CONFORMING-CODE-SEGMENT;
        GO TO NONCONFORMING-CODE-SEGMENT;
        GO TO CALL-GATE;
        GO TO TASK-GATE;
        GO TO TASK-STATE-SEGMENT;
    FI;
  FI;

```

```

CONFORMING-CODE-SEGMENT:
  IF L bit = 1 and D bit = 1 and IA32_EFER.LMA = 1
    THEN GP(new code segment selector); FI;
  IF DPL > CPL
    THEN #GP(new code segment selector); FI;
  IF CR4.FRED = 1 and CPL = 0 and L bit = 0

```



```

    THEN GP(new code segment selector); FI;
IF segment not present
    THEN #NP(new code segment selector); FI;
IF stack not large enough for return address
    THEN #SS(0); FI;
tempEIP := DEST(Offset);
IF target mode = Compatibility mode
    THEN tempEIP := tempEIP AND 00000000_FFFFFFFFH; FI;
IF OperandSize = 16
    THEN
        tempEIP := tempEIP AND 0000FFFFH; FI; (* Clear upper 16 bits *)
IF (IA32_EFER.LMA = 0 or target mode = Compatibility mode) and (tempEIP outside new code segment limit)
    THEN #GP(0); FI;
IF tempEIP is non-canonical
    THEN #GP(0); FI;
IF ShadowStackEnabled(CPL)
    IF OperandSize = 32
        THEN
            tempPushLIP = CSBASE + EIP;
        ELSE
            IF OperandSize = 16
                THEN
                    tempPushLIP = CSBASE + IP;
                ELSE (* OperandSize = 64 *)
                    tempPushLIP = RIP;
            FI;
        FI;
    tempPushCS = CS;
FI;
IF OperandSize = 32
    THEN
        Push(CS); (* Padded with 16 high-order bits *)
        Push(EIP);
        CS := DEST(CodeSegmentSelector);
        (* Segment descriptor information also loaded *)
        CS(RPL) := CPL;
        EIP := tempEIP;
    ELSE
        IF OperandSize = 16
            THEN
                Push(CS);
                Push(IP);
                CS := DEST(CodeSegmentSelector);
                (* Segment descriptor information also loaded *)
                CS(RPL) := CPL;
                EIP := tempEIP;
            ELSE (* OperandSize = 64 *)
                Push(CS); (* Padded with 48 high-order bits *)
                Push(RIP);
                CS := DEST(CodeSegmentSelector);
                (* Segment descriptor information also loaded *)
                CS(RPL) := CPL;
                RIP := tempEIP;
            FI;
        FI;

```

```

FI;
IF ShadowStackEnabled(CPL)
    IF (IA32_EFER.LMA and DEST(CodeSegmentSelector).L) = 0
        (* If target is legacy or compatibility mode then the SSP must be in low 4GB *)
        IF (SSP & 0xFFFFFFFF00000000 != 0)
            THEN #GP(0); FI;

        FI;
        (* align to 8 byte boundary if not already aligned *)
        tempSSP = SSP;
        Shadow_stack_store 4 bytes of 0 to (SSP - 4)
        SSP = SSP & 0xFFFFFFFFFFFFFFF8H
        ShadowStackPush8B(tempPushCS); (* Padded with 48 high-order bits of 0 *)
        ShadowStackPush8B(tempPushLIP); (* Padded with 32 high-order bits of 0 for 32 bit LIP*)
        ShadowStackPush8B(tempSSP);
    FI;
IF EndbranchEnabled(CPL)
    IF CPL = 3
        THEN
            IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_U_CET.SUPPRESS = 0
        ELSE
            IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_S_CET.SUPPRESS = 0
        FI;
FI;
END;

NONCONFORMING-CODE-SEGMENT:
    IF L-Bit = 1 and D-BIT = 1 and IA32_EFER.LMA = 1
        THEN GP(new code segment selector); FI;
    IF (RPL > CPL) or (DPL ≠ CPL)
        THEN #GP(new code segment selector); FI;
    IF CR4.FRED = 1 and CPL = 0 and L bit = 0
        THEN GP(new code segment selector); FI;
    IF segment not present
        THEN #NP(new code segment selector); FI;
    IF stack not large enough for return address
        THEN #SS(0); FI;
    tempEIP := DEST(Offset);
    IF target mode = Compatibility mode
        THEN tempEIP := tempEIP AND 00000000_FFFFFFFFH; FI;
    IF OperandSize = 16
        THEN tempEIP := tempEIP AND 0000FFFFH; FI; (* Clear upper 16 bits *)
    IF (IA32_EFER.LMA = 0 or target mode = Compatibility mode) and (tempEIP outside new code segment limit)
        THEN #GP(0); FI;
    IF tempEIP is non-canonical
        THEN #GP(0); FI;
    IF ShadowStackEnabled(CPL)
        IF IA32_EFER.LMA & CS.L
            tempPushLIP = RIP
        ELSE
            tempPushLIP = CSBASE + EIP;
        FI;
    tempPushCS = CS;

```

```

FI;
IF OperandSize = 32
    THEN
        Push(CS); (* Padded with 16 high-order bits *)
        Push(EIP);
        CS := DEST(CodeSegmentSelector);
        (* Segment descriptor information also loaded *)
        CS(RPL) := CPL;
        EIP := tempEIP;
    ELSE
        IF OperandSize = 16
            THEN
                Push(CS);
                Push(IP);
                CS := DEST(CodeSegmentSelector);
                (* Segment descriptor information also loaded *)
                CS(RPL) := CPL;
                EIP := tempEIP;
            ELSE (* OperandSize = 64 *)
                Push(CS); (* Padded with 48 high-order bits *)
                Push(RIP);
                CS := DEST(CodeSegmentSelector);
                (* Segment descriptor information also loaded *)
                CS(RPL) := CPL;
                RIP := tempEIP;
            FI;
        FI;
FI;
IF ShadowStackEnabled(CPL)
    IF (IA32_EFER.LMA and DEST(CodeSegmentSelector).L) = 0
        (* If target is legacy or compatibility mode then the SSP must be in low 4GB *)
        IF (SSP & 0xFFFFFFFF00000000 != 0)
            THEN #GP(0); FI;
        FI;
    (* align to 8 byte boundary if not already aligned *)
    tempSSP = SSP;
    Shadow_stack_store 4 bytes of 0 to (SSP - 4)
    SSP = SSP & 0xFFFFFFFFFFFFFFF8H
    ShadowStackPush8B(tempPushCS); (* Padded with 48 high-order 0 bits *)
    ShadowStackPush8B(tempPushLIP); (* Padded 32 high-order bits of 0 for 32 bit LIP*)
    ShadowStackPush8B(tempSSP);
    FI;
IF EndbranchEnabled(CPL)
    IF CPL = 3
        THEN
            IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_U_CET.SUPPRESS = 0
        ELSE
            IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_S_CET.SUPPRESS = 0
        FI;
    FI;
FI;
END;

```

CALL-GATE:

```

IF call gate (DPL < CPL) or (RPL > DPL) or (CR4.FRED = 1)
    THEN #GP(call-gate selector); FI;
IF call gate not present
    THEN #NP(call-gate selector); FI;
IF call-gate code-segment selector is NULL
    THEN #GP(0); FI;
IF call-gate code-segment selector index is outside descriptor table limits
    THEN #GP(call-gate code-segment selector); FI;
Read call-gate code-segment descriptor;
IF call-gate code-segment descriptor does not indicate a code segment
or call-gate code-segment descriptor DPL > CPL
    THEN #GP(call-gate code-segment selector); FI;
IF IA32_EFER.LMA = 1 AND (call-gate code-segment descriptor is
not a 64-bit code segment or call-gate code-segment descriptor has both L-bit and D-bit set)
    THEN #GP(call-gate code-segment selector); FI;
IF call-gate code segment not present
    THEN #NP(call-gate code-segment selector); FI;
IF call-gate code segment is non-conforming and DPL < CPL
    THEN go to MORE-PRIVILEGE;
    ELSE go to SAME-PRIVILEGE;
FI;
END;

```

MORE-PRIVILEGE:

```

IF current TSS is 32-bit
    THEN
        TSSstackAddress := (new code-segment DPL * 8) + 4;
        IF (TSSstackAddress + 5) > current TSS limit
            THEN #TS(current TSS selector); FI;
        NewSS := 2 bytes loaded from (TSS base + TSSstackAddress + 4);
        NewESP := 4 bytes loaded from (TSS base + TSSstackAddress);
    ELSE
        IF current TSS is 16-bit
            THEN
                TSSstackAddress := (new code-segment DPL * 4) + 2
                IF (TSSstackAddress + 3) > current TSS limit
                    THEN #TS(current TSS selector); FI;
                NewSS := 2 bytes loaded from (TSS base + TSSstackAddress + 2);
                NewESP := 2 bytes loaded from (TSS base + TSSstackAddress);
            ELSE (* current TSS is 64-bit *)
                TSSstackAddress := (new code-segment DPL * 8) + 4;
                IF (TSSstackAddress + 7) > current TSS limit
                    THEN #TS(current TSS selector); FI;
                NewSS := new code-segment DPL; (* NULL selector with RPL = new CPL *)
                NewRSP := 8 bytes loaded from (current TSS base + TSSstackAddress);
        FI;
    FI;
IF IA32_EFER.LMA = 0 and NewSS is NULL
    THEN #TS(NewSS); FI;
Read new stack-segment descriptor;
IF IA32_EFER.LMA = 0 and (NewSS RPL ≠ new code-segment DPL
or new stack-segment DPL ≠ new code-segment DPL or new stack segment is not a
writable data segment)
    THEN #TS(NewSS); FI

```

```

IF IA32_EFER.LMA = 0 and new stack segment not present
    THEN #SS(NewSS); FI;
IF CallGateSize = 32
    THEN
        IF new stack does not have room for parameters plus 16 bytes
            THEN #SS(NewSS); FI;
        IF CallGate(InstructionPointer) not within new code-segment limit
            THEN #GP(0); FI;
        SS := newSS; (* Segment descriptor information also loaded *)
        ESP := newESP;
        CS:EIP := CallGate(CS:InstructionPointer);
        (* Segment descriptor information also loaded *)
        Push(oldSS:oldESP); (* From calling procedure *)
        temp := parameter count from call gate, masked to 5 bits;
        Push(parameters from calling procedure's stack, temp)
        Push(oldCS:oldEIP); (* Return address to calling procedure *)
    ELSE
        IF CallGateSize = 16
            THEN
                IF new stack does not have room for parameters plus 8 bytes
                    THEN #SS(NewSS); FI;
                IF (CallGate(InstructionPointer) AND FFFFH) not in new code-segment limit
                    THEN #GP(0); FI;
                SS := newSS; (* Segment descriptor information also loaded *)
                ESP := newESP;
                CS:IP := CallGate(CS:InstructionPointer);
                (* Segment descriptor information also loaded *)
                Push(oldSS:oldESP); (* From calling procedure *)
                temp := parameter count from call gate, masked to 5 bits;
                Push(parameters from calling procedure's stack, temp)
                Push(oldCS:oldEIP); (* Return address to calling procedure *)
            ELSE (* CallGateSize = 64 *)
                IF pushing 32 bytes on the stack would use a non-canonical address
                    THEN #SS(NewSS); FI;
                IF (CallGate(InstructionPointer) is non-canonical)
                    THEN #GP(0); FI;
                SS := NewSS; (* NewSS is NULL)
                RSP := NewESP;
                CS:IP := CallGate(CS:InstructionPointer);
                (* Segment descriptor information also loaded *)
                Push(oldSS:oldESP); (* From calling procedure *)
                Push(oldCS:oldEIP); (* Return address to calling procedure *)
            FI;
        FI;
    IF ShadowStackEnabled(CPL) AND CPL = 3
        THEN
            IF IA32_EFER.LMA = 0
                THEN IA32_PL3_SSP := SSP;
            ELSE (* adjust so bits 63:N get the value of bit N-1, where N is the CPU's maximum linear-address width *)
                IA32_PL3_SSP := LA_adjust(SSP);
            FI;
        FI;
    CPL := CodeSegment(DPL)
    CS(RPL) := CPL

```

```

IF ShadowStackEnabled(CPL)
  oldSSP := SSP
  SSP := IA32_PLI_SSP; (* where i is the CPL *)
  IF SSP & 0x07 != 0 (* if SSP not aligned to 8 bytes then #GP *)
    THEN #GP(0); FI;
  (* Token and CS:LIP:oldSSP pushed on shadow stack must be contained in a naturally aligned 32-byte region*)
  IF (SSP & ~0x1F) != ((SSP - 24) & ~0x1F)
    #GP(0); FI;
  IF ((IA32_EFER.LMA and CS.L) = 0 AND SSP[63:32] != 0)
    THEN #GP(0); FI;
  expected_token_value = SSP          (* busy bit - bit position 0 - must be clear *)
  new_token_value = SSP | BUSY_BIT    (* Set the busy bit *)
  IF shadow_stack_lock_cmpxchg8b(SSP, new_token_value, expected_token_value) != expected_token_value
    THEN #GP(0); FI;
  IF oldSS.DPL != 3
    ShadowStackPush8B(oldCS); (* Padded with 48 high-order bits of 0 *)
    ShadowStackPush8B(oldCSBASE+oldRIP); (* Padded with 32 high-order bits of 0 for 32 bit LIP*)
    ShadowStackPush8B(oldSSP);
  FI;
FI;
IF EndbranchEnabled (CPL)
  IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
  IA32_S_CET.SUPPRESS = 0
FI;
END;

```

SAME-PRIVILEGE:

```

IF CallGateSize = 32
  THEN
    IF stack does not have room for 8 bytes
      THEN #SS(0); FI;
    IF CallGate(InstructionPointer) not within code segment limit
      THEN #GP(0); FI;
    CS:EIP := CallGate(CS:EIP) (* Segment descriptor information also loaded *)
    Push(oldCS:oldEIP); (* Return address to calling procedure *)
  ELSE
    If CallGateSize = 16
      THEN
        IF stack does not have room for 4 bytes
          THEN #SS(0); FI;
        IF CallGate(InstructionPointer) not within code segment limit
          THEN #GP(0); FI;
        CS:IP := CallGate(CS:instruction pointer);
        (* Segment descriptor information also loaded *)
        Push(oldCS:oldIP); (* Return address to calling procedure *)
      ELSE (* CallGateSize = 64)
        IF pushing 16 bytes on the stack touches non-canonical addresses
          THEN #SS(0); FI;
        IF RIP non-canonical
          THEN #GP(0); FI;
        CS:IP := CallGate(CS:instruction pointer);
        (* Segment descriptor information also loaded *)
        Push(oldCS:oldIP); (* Return address to calling procedure *)
      FI;
  FI;

```

```

FI;
CS(RPL) := CPL
IF ShadowStackEnabled(CPL)
    (* Align to next 8 byte boundary *)
    tempSSP = SSP;
    Shadow_stack_store 4 bytes of 0 to (SSP - 4)
    SSP = SSP & 0xFFFFFFFFFFFFF8H;
    (* push cs:lip:ssp on shadow stack *)
    ShadowStackPush8B(oldCS); (* Padded with 48 high-order bits of 0 *)
    ShadowStackPush8B(oldCSBASE + oldRIP); (* Padded with 32 high-order bits of 0 for 32 bit LIP*)
    ShadowStackPush8B(tempSSP);
FI;
IF EndbranchEnabled (CPL)
    IF CPL = 3
        THEN
            IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH;
            IA32_U_CET.SUPPRESS = 0
        ELSE
            IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH;
            IA32_S_CET.SUPPRESS = 0
    FI;
FI;
END;

```

TASK-GATE:

```

IF task gate DPL < CPL or RPL
    THEN #GP(task gate selector); FI;
IF task gate not present
    THEN #NP(task gate selector); FI;
Read the TSS segment selector in the task-gate descriptor;
IF TSS segment selector local/global bit is set to local
or index not within GDT limits
    THEN #GP(TSS selector); FI;
Access TSS descriptor in GDT;
IF descriptor is not a TSS segment
    THEN #GP(TSS selector); FI;
IF TSS descriptor specifies that the TSS is busy
    THEN #GP(TSS selector); FI;
IF TSS not present
    THEN #NP(TSS selector); FI;
SWITCH-TASKS (with nesting) to TSS;
IF EIP not within code segment limit
    THEN #GP(0); FI;
END;

```

TASK-STATE-SEGMENT:

```

IF TSS DPL < CPL or RPL
or TSS descriptor indicates TSS not available
    THEN #GP(TSS selector); FI;
IF TSS is not present
    THEN #NP(TSS selector); FI;
SWITCH-TASKS (with nesting) to TSS;
IF EIP not within code segment limit
    THEN #GP(0); FI;

```

END;

Flags Affected

All flags are affected if a task switch occurs; no flags are affected if a task switch does not occur.

Protected Mode Exceptions

#GP(0)	If the target offset in destination operand is beyond the new code segment limit. If the segment selector in the destination operand is NULL. If the code segment selector in the gate is NULL. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector. If target mode is compatibility mode and SSP is not in low 4GB. If SSP in IA32_PLI_SSP (where i is the new CPL) is not 8 byte aligned. If the token and the stack frame to be pushed on shadow stack are not contained in a naturally aligned 32-byte region of the shadow stack. If “supervisor Shadow Stack” token on new shadow stack is marked busy. If destination mode is 32-bit or compatibility mode, but SSP address in “supervisor shadow stack” token is beyond 4GB. If SSP address in “supervisor shadow stack” token does not match SSP address in IA32_PLI_SSP (where i is the new CPL).
#GP(selector)	If a code segment or gate or TSS selector index is outside descriptor table limits. If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, call gate, task gate, or task state segment. If the DPL for a nonconforming-code segment is not equal to the CPL or the RPL for the segment’s segment selector is greater than the CPL. If the DPL for a conforming-code segment is greater than the CPL. If the DPL from a call-gate, task-gate, or TSS segment descriptor is less than the CPL or than the RPL of the call-gate, task-gate, or TSS’s segment selector. If the segment descriptor for a segment selector from a call gate does not indicate it is a code segment. If the segment selector from a call gate is beyond the descriptor table limits. If the DPL for a code-segment obtained from a call gate is greater than the CPL. If the segment selector for a TSS has its local/global bit set for local. If a TSS segment descriptor specifies that the TSS is busy or not available.
#SS(0)	If pushing the return address, parameters, or stack segment pointer onto the stack exceeds the bounds of the stack segment, when no stack switch occurs. If a memory operand effective address is outside the SS segment limit.
#SS(selector)	If pushing the return address, parameters, or stack segment pointer onto the stack exceeds the bounds of the stack segment, when a stack switch occurs. If the SS register is being loaded as part of a stack switch and the segment pointed to is marked not present. If stack segment does not have room for the return address, parameters, or stack segment pointer, when stack switch occurs.
#NP(selector)	If a code segment, data segment, call gate, task gate, or TSS is not present.

#TS(selector)	If the new stack segment selector and ESP are beyond the end of the TSS. If the new stack segment selector is NULL. If the RPL of the new stack segment selector in the TSS is not equal to the DPL of the code segment being accessed. If DPL of the stack segment descriptor for the new stack segment is not equal to the DPL of the code segment descriptor. If the new stack segment is not a writable data segment. If segment-selector index for stack segment is outside descriptor table limits.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the target offset is beyond the code segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the target offset is beyond the code segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

#GP(selector)	If a memory address accessed by the selector is in non-canonical space. <i>If FRED transitions are enabled and the selector references a call gate.</i>
#GP(0)	If the target offset in the destination operand is non-canonical.

64-Bit Mode Exceptions

#GP(0)	If a memory address is non-canonical. If target offset in destination operand is non-canonical. If the segment selector in the destination operand is NULL. If the code segment selector in the 64-bit gate is NULL. If target mode is compatibility mode and SSP is not in low 4GB. If SSP in IA32_PLi_SSP (where i is the new CPL) is not 8 byte aligned. If the token and the stack frame to be pushed on shadow stack are not contained in a naturally aligned 32-byte region of the shadow stack. If “supervisor Shadow Stack” token on new shadow stack is marked busy. If destination mode is 32-bit mode or compatibility mode, but SSP address in “super-visor shadow” stack token is beyond 4GB. If SSP address in “supervisor shadow stack” token does not match SSP address in IA32_PLi_SSP (where i is the new CPL).

#GP(selector)	If code segment or 64-bit call gate is outside descriptor table limits. If code segment or 64-bit call gate overlaps non-canonical space. If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, or 64-bit call gate. If the segment descriptor pointed to by the segment selector in the destination operand is a code segment and has both the D-bit and the L- bit set. If the DPL for a nonconforming-code segment is not equal to the CPL, or the RPL for the segment's segment selector is greater than the CPL. If the DPL for a conforming-code segment is greater than the CPL. If the DPL from a 64-bit call-gate is less than the CPL or than the RPL of the 64-bit call-gate. If the upper type field of a 64-bit call gate is not 0x0. If the segment selector from a 64-bit call gate is beyond the descriptor table limits. If the DPL for a code-segment obtained from a 64-bit call gate is greater than the CPL. If the code segment descriptor pointed to by the selector in the 64-bit gate doesn't have the L-bit set and the D-bit clear. If the segment descriptor for a segment selector from the 64-bit call gate does not indicate it is a code segment. If FRED transitions are enabled and the selector references a call gate. If FRED transitions are enabled, CPL is 0, and the instruction would enter compatibility mode.
#SS(0)	If pushing the return offset or CS selector onto the stack exceeds the bounds of the stack segment when no stack switch occurs. If a memory operand effective address is outside the SS segment limit. If the stack address is in a non-canonical form.
#SS(selector)	If pushing the old values of SS selector, stack pointer, EFLAGS, CS selector, offset, or error code onto the stack violates the canonical boundary when a stack switch occurs.
#NP(selector)	If a code segment or 64-bit call gate is not present.
#TS(selector)	If the load of the new RSP exceeds the limit of the TSS.
#UD	(64-bit mode only) If a far call is direct to an absolute address in memory. If the LOCK prefix is used.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

CLRSSBSY—Clear Busy Flag in a Supervisor Shadow Stack Token

Opcode/ Instruction	Op / En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F AE /6 CLRSSBSY m64	M	V/V	CET_SS	Clear busy flag in supervisor shadow stack token reference by m64.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
M	N/A	ModRM:r/m (r, w)	N/A	N/A	N/A

Description

Clear busy flag in supervisor shadow stack token reference by m64. Subsequent to marking the shadow stack as not busy the SSP is loaded with value 0.

This instruction cannot be executed when FRED transitions are enabled. FRED transitions do not use supervisor shadow stack tokens.

Operation

IF CR4.CET = 0 OR CR4.FRED = 1
THEN #UD; FI;

IF IA32_S_CET.SH_STK_EN = 0
THEN #UD; FI;

IF CPL > 0
THEN GP(0); FI;

SSP_LA = Linear_Address(mem operand)

IF SSP_LA not aligned to 8 bytes
THEN #GP(0); FI;

expected_token_value = SSP_LA | BUSY_BIT (* busy bit - bit position 0 - must be set *)

new_token_value = SSP_LA (* Clear the busy bit *)

IF shadow_stack_lock_cmpxchg8b(SSP_LA, new_token_value, expected_token_value) != expected_token_value
invalid_token := 1; FI

(* Set the CF if invalid token was detected *)

RFLAGS.CF = (invalid_token == 1) ? 1 : 0;

RFLAGS.ZF,PF,AF,OF,SF := 0;

SSP := 0

Flags Affected

CF is set if an invalid token was detected, else it is cleared. ZF, PF, AF, OF, and SF are cleared.

Protected Mode Exceptions

#UD	If the LOCK prefix is used. If CR4.CET = 0. If IA32_S_CET.SH_STK_EN = 0.
#GP(0)	If memory operand linear address not aligned to 8 bytes. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If destination is located in a non-writable segment. If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector. If CPL is not 0.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.

Real-Address Mode Exceptions

#UD	The CLRSSBSY instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The CLRSSBSY instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	If the LOCK prefix is used. If CR4.CET = 0. If IA32_S_CET.SH_STK_EN = 0. If CR4.FRED = 1.
#GP(0)	If memory operand linear address not aligned to 8 bytes. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If destination is located in a non-writable segment. If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector. If CPL is not 0.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.

64-Bit Mode Exceptions

#UD	If the LOCK prefix is used. If CR4.CET = 0. If CR4.FRED = 1. If IA32_S_CET.SH_STK_EN = 0.
#GP(0)	If memory operand linear address not aligned to 8 bytes. If CPL is not 0. If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.

CVTTSD2SI—Convert With Truncation Scalar Double Precision Floating-Point Value to Signed Integer

Opcode/ Instruction	Op / En	64/32 bit Mode Support	CPUID Feature Flag	Description
F2 OF 2C /r CVTTSD2SI r32, xmm1/m64	A	V/V	SSE2	Convert one double precision floating-point value from xmm1/m64 to one signed doubleword integer in r32 using truncation.
F2 REX.W OF 2C /r CVTTSD2SI r64, xmm1/m64	A	V/N.E.	SSE2	Convert one double precision floating-point value from xmm1/m64 to one signed quadword integer in r64 using truncation.
VEX.LIG.F2.OF.W0 2C /r ¹ VCVTTSD2SI r32, xmm1/m64	A	V/V	AVX	Convert one double precision floating-point value from xmm1/m64 to one signed doubleword integer in r32 using truncation.
VEX.LIG.F2.OF.W1 2C /r ¹ VCVTTSD2SI r64, xmm1/m64	B	V/N.E. ²	AVX	Convert one double precision floating-point value from xmm1/m64 to one signed quadword integer in r64 using truncation.
EVEX.LLIG.F2.OF.W0 2C /r VCVTTSD2SI r32, xmm1/m64{sae}	B	V/V	AVX512F OR AVX10.1	Convert one double precision floating-point value from xmm1/m64 to one signed doubleword integer in r32 using truncation.
EVEX.LLIG.F2.OF.W1 2C /r VCVTTSD2SI r64, xmm1/m64{sae}	B	V/N.E. ²	AVX512F OR AVX10.1	Convert one double precision floating-point value from xmm1/m64 to one signed quadword integer in r64 using truncation.

NOTES:

1. Software should ensure VCVTTSD2SI is encoded with VEX.L=0. Encoding VCVTTSD2SI with VEX.L=1 may encounter unpredictable behavior across different processor generations.
2. For this specific instruction, VEX.W/EVEX.W in non-64 bit is ignored; the instructions behaves as if the W0 version is used.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A
B	Tuple1 Fixed	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

Converts a double precision floating-point value in the source operand (the second operand) to a signed doubleword integer (or signed quadword integer if operand size is 64 bits) in the destination operand (the first operand). The source operand can be an XMM register or a 64-bit memory location. The destination operand is a general purpose register. When the source operand is an XMM register, the double precision floating-point value is contained in the low quadword of the register.

When a conversion is inexact, a **truncated (round toward zero) result is returned**.

If a converted result exceeds the range limits of signed doubleword integer (in non-64-bit modes or 64-bit mode with REX.W/VEX.W/EVEX.W=0), the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value 80000000H is returned.

If a converted result exceeds the range limits of signed quadword integer (in 64-bit mode and REX.W/VEX.W/EVEX.W = 1), the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value 80000000_00000000H is returned.

Legacy SSE instructions: In 64-bit mode, Use of the REX.W prefix promotes the instruction to 64-bit operation. See the summary chart at the beginning of this section for encoding data and limits.

VEX.W1 and EVEX.W1 versions: promotes the instruction to produce 64-bit data in 64-bit mode.

Note: VEX.vvvv and EVEX.vvvv are reserved and must be 1111b, otherwise instructions will #UD.
Software should ensure VCVTTSD2SI is encoded with VEX.L=0. Encoding VCVTTSD2SI with VEX.L=1 may encounter unpredictable behavior across different processor generations.

Operation

(V)CVTTSD2SI (All Versions)

IF 64-Bit Mode and OperandSize = 64

THEN

DEST[63:0] := Convert_Double_Precision_Floating_Point_To_Integer_Truncate(SRC[63:0]);

ELSE

DEST[31:0] := Convert_Double_Precision_Floating_Point_To_Integer_Truncate(SRC[63:0]);

FI;

Intel C/C++ Compiler Intrinsic Equivalent

VCVTTSD2SI int __mm_cvttssd_i32(__m128d a);

VCVTTSD2SI int __mm_cvtt_roundssd_i32(__m128d a, int sae);

VCVTTSD2SI __int64 __mm_cvttssd_i64(__m128d a);

VCVTTSD2SI __int64 __mm_cvtt_roundssd_i64(__m128d a, int sae);

CVTTSD2SI int __mm_cvttssd_si32(__m128d a);

CVTTSD2SI __int64 __mm_cvttssd_si64(__m128d a);

SIMD Floating-Point Exceptions

Invalid, Precision.

Other Exceptions

VEX-encoded instructions, see Table 2-20, “Type 3 Class Exception Conditions,” additionally:

#UD If VEX.vvvv != 1111B.

EVEX-encoded instructions, see Table 2-50, “Type E3NF Class Exception Conditions.”

ERETS—Event Return to Supervisor

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
F2 0F 01 CA	ERETS	Z0	Valid	Invalid	Event return for events occurring in ring 0.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

ERETS returns from an event handler, establishing the state based on the contents of the stack (typically, that which was in effect before FRED event delivery). ERETS can be executed only if CPL = 0, and it does not change CPL. For this reason, ERETS is used to return from handling events that occurred while CPL = 0.

ERETS takes no explicit arguments; its operation depends on the contents of the regular stack and (when enabled) the shadow stack.

Execution of ERETS causes an invalid-opcode exception (#UD) if FRED transitions are not enabled or if CPL > 0. For this reason, ERETS can be executed only in 64-bit mode.

Section 8.4.1, “ERETS (Event Return to Supervisor),” of the Intel® 64 and IA-32 Architectures software developer’s Manual, Volume 3 includes a detailed discussion of ERETS.

Instruction ordering. Instructions following execution of ERETS may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to ERETS have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Operation

IF CR4.FRED = 0 OR CPL > 0

THEN #UD;

FI;

// CR4.FRED = 1 and CPL = 0 implies IA32_EFER.LMA = CS.L = 1

// pop old state from regular stack and check it

RSP := RSP + 8; // skip over error code so that RSP references the return state

pop8B newRIP;

pop8B tempCS; // not used to load CS

pop8B newRFLAGS;

pop8B newRSP;

pop8B tempSS; // not used to load SS

IF newRIP is not paging canonical OR

tempCS & FFFFFFFF_FFF8FFFFH ≠ current CS selector OR

newRFLAGS & FFFFFFFF_FFC2802AH ≠ 2 OR // enforce bit 1 set; VM, reserved bits clear

tempSS & FFF8FFFFH ≠ current SS selector OR // do not check bits 63:32

THEN #GP(0);

FI;

// ERETS will not numerically increase stack level

newCSL := min[CSL, tempCS[17:16]];

IBT_restore := tempCS[18];

STI_block := tempSS[16];

pend_DB := tempSS[17];

NMI_unblock := tempSS[18];

```

// If supervisor shadow stacks are enabled, pop and check values from the shadow stack
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1
    THEN
        IF SSP & 7 ≠ 0                                // require 8-byte alignment
            THEN #CP(FAR-RET/IRET);
        FI;
        popSS_8B newSSP;
        popSS_8B checkSSLIP;
        popSS_8B checkSSCS;
        IF checkSSCS ≠ tempCS                        // 64-bit compare
            OR checkSSLIP ≠ newRIP
            OR newSSP & 3H ≠ 0
            THEN #CP(FAR-RET/IRET);
        FI;
        IF newSSP not CPU canonical
            THEN #GP(0);
        FI;
        // If the stack level is changing, compare SSP to the FRED SSP MSR for the old stack level
        IF newCSL < CSL AND IA32_FRED_SSPi ≠ SSP // where i = CSL
            THEN #CP(FAR-RET/IRET);
        FI;
    FI;

// update registers for return state
RIP := newRIP;
RFLAGS := newRFLAGS;                                // ERETS can set RFLAGS.RF to 1
RSP := newRSP;
CSL := newCSL;                                        // reflect in IA32_FRED_CONFIG[1:0]
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1
    THEN SSP := newSSP;
FI;
IF CR4.CET = 1 AND IA32_S_CET.ENDBR_EN = 1 AND IA32_S_CET.SUPPRESS = 0 AND IBT_restore = 1
    THEN IA32_S_CET.TRACKER := 1;
FI;

// update event-related state
IF STI_block = 1 AND RFLAGS.IF = 1 AND STI blocking was not in effect prior to ERETS
    THEN establish STI blocking after ERETS;
FI;
IF pend_DB = 1 AND RFLAGS.TF = 1
    THEN pend a single-step debug exception (#DB) to be delivered after ERETS;
FI;
IF NMI_unblock = 1
    THEN unblock NMIs;
FI;

```

Flags Affected

All defined flags and fields in the RFLAGS register are potentially modified except for the VM flag.

Protected Mode Exceptions

#UD The ERETS instruction is not recognized in protected mode.

Real-Address Mode Exceptions

#UD The ERETS instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The ERETS instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD The ERETS instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD If CR4.FRED = 0.
If CPL > 0.
If the LOCK prefix is used.

#GP(0) If the RIP image on the stack is a not paging canonical.
If the RFLAGS image on the stack sets reserved bits or the VM bit (or fails to set bit 1).
If bits 15:0 of the CS image on the stack differs from the value of the current CS selector.
If the CS image on the stack sets undefined bits (bits 63:19).
If bits 15:0 of the SS image on the stack differs from the value of the current SS selector.
If the SS image on the stack sets undefined bits (bits 31:19).
If the SSP image on the shadow stack is not CPU canonical.
If a shadow-stack access would use an address that is not paging canonical or would cause a LASS violation.

#SS(0) If an ordinary stack access would use an address that is not paging canonical or would cause a LASS violation.

#CP(FAR-RET/IRET) If the address in SSP is not 8-byte aligned.
If the 64-bit CS image on the shadow stack differs from that on the regular stack.
If the SSLIP image on the shadow stack differs from the RIP image on the regular stack.
If the SSP image on the shadow stack is not 4-byte aligned.
If the CSL is i, the stack level is changing, and SSP differs from the value of IA32_FRED_SSPI.

ERETU—Event Return to User

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
F3 0F 01 CA	ERETU	Z0	Valid	Valid	Event return for events occurring in ring 3.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

ERETU returns from an event handler, establishing the state based on the contents of the stack (typically, that which was in effect before FRED event delivery). ERETU can be executed only if CPL = 0, and it changes CPL to 3. For this reason, ERETU is used to return from handling events that occurred while CPL = 3.

ERETU takes no explicit arguments; its operation depends on the contents of the regular stack.

Execution of ERETU causes an invalid-opcode exception (#UD) if FRED transitions are not enabled or if CPL > 0. For this reason, ERETU can be executed only in 64-bit mode.

ERETU establishes new values of CS and SS in one of three different ways:

- If the values popped from the stack correspond to the values of IA32_STAR[63:48] + 16 and IA32_STAR[63:48] + 8, respectively, CS and SS are loaded with standard values for operation in 64-bit mode (similar those established by the 64-bit form of SYSCALL).
- If the values popped from the stack correspond to the values of IA32_STAR[63:48] and IA32_STAR[63:48] + 8, respectively, CS and SS are loaded with standard values for operation in compatibility mode (similar those established by the 32-bit form of SYSCALL).
- Otherwise, CS and SS are loaded from the GDT or LDT using the selectors popped from the stack (similar to the manner used by the 64-bit form of IRET).

For further details see the Operation section below and Section 8.4.2, “ERETU (Event Return to User),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3.

Instruction ordering. Instructions following execution of ERETU may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to ERETU have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Operation

```
IF CR4.FRED = 0 OR CS.L = 0 OR CPL > 0
    THEN #UD;
```

```
FI;
```

```
IF CSL > 0
    THEN #GP(0);
```

```
FI;
```

```
// pop old state from regular stack and check it
```

```
RSP := RSP + 8;           // skip over error code so that RSP references the return state
```

```
pop8B newRIP;
```

```
pop8B tempCS;
```

```
pop8B newRFLAGS;
```

```
pop8B newRSP;
```

```
pop8B tempSS;
```

```
IF tempCS & FFFFFFFF_FFFF0003H ≠ 3 OR           // enforce return to ring 3
```

```
    newRFLAGS & FFFFFFFF_FFC2B02AH ≠ 2 OR       // enforce bit 1 set; IOPL, VM, reserved bits clear
```

```
    tempSS & FFF80003H ≠ 3                       // do not check bits 63:32
```

```
    THEN #GP(0);
```

```

FI;
pend_DB := tempSS[17];
NMI_unblock := tempSS[18];
IF tempCS[15:0] = IA32_STAR[63:48] + 16 AND tempSS[15:0] = IA32_STAR[63:48] + 8
    THEN                                     // Return to ring 3 in standard 64-bit configuration
        // set newCS to standard values used ring 3 in 64-bit mode
        newCS.selector := tempCS[15:0];
        newCS.base := 0;
        newCS.limit := FFFFFFFH;
        newCS.type := 11;
        newCS.S := 1;
        newCS.DPL := 3;
        newCS.P := 1;
        newCS.L := 1;
        newCS.D := 0;
        newCS.G := 1;
        newCS.unusable := 0;
        // set newSS to standard values for ring 3
        newSS.selector := tempSS[15:0];
        newSS.base := 0;
        newSS.limit := FFFFFFFH;
        newSS.type := 3;
        newSS.S := 1;
        newSS.DPL := 3;
        newSS.P := 1;
        newSS.B := 1;
        newSS.G := 1;
        newSS.unusable := 0;
    ELSIF tempCS[15:0] = IA32_STAR[63:48] AND tempSS[15:0] = IA32_STAR[63:48] + 8
    THEN
        // set newCS to standard values used ring 3 in compatibility mode
        newCS.selector := tempCS[15:0];
        newCS.base := 0;
        newCS.limit := FFFFFFFH;
        newCS.type := 11;
        newCS.S := 1;
        newCS.DPL := 3;
        newCS.P := 1;
        newCS.L := 0;
        newCS.D := 1;
        newCS.G := 1;
        newCS.unusable := 0;
        // set newSS to standard values for ring 3
        newSS.selector := tempSS[15:0];
        newSS.base := 0;
        newSS.limit := FFFFFFFH;
        newSS.type := 3;
        newSS.S := 1;
        newSS.DPL := 3;
        newSS.P := 1;
        newSS.B := 1;
        newSS.G := 1;
        newSS.unusable := 0;
    ELSE

```

```

        load newCS using tempCS[15:0];           // load each as is done by IRET, including
        load newSS using tempSS[15:0];           // checks that may lead to a fault
FI;
IF newCS.L = 1
    THEN // return to 64-bit mode
        IF newRIP is not paging canonical
            THEN #GP(0);
    FI;
ELSE // return to compatibility mode
    newRIP[63:32] := 0;
    IF newRIP is not within newCS's limit (based on limit field and G bit)
        // newRIP is always within the limit with standard values for ring 3 in compatibility mode
        THEN #GP(0);
    FI;
    newRSP[63:32] := 0;
FI;
// If user shadow stacks are enabled, check new SSP value on return to compatibility mode
IF CR4.CET = 1 AND IA32_U_CET.SH_STK_EN = 1 AND newCS.L = 0 AND IA32_PL3_SSP[63:32] ≠ 0
    THEN #GP(0);
FI;
// If supervisor shadow stacks are enabled, compare SSP to the FRED SSP MSR for stack level 0
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1 AND IA32_FRED_SSP0 ≠ SSP
    THEN #CP(FAR-RET/IRET);
FI;
// update registers for return state
RIP := newRIP;
RFLAGS := newRFLAGS;           // ERETU can set RFLAGS.RF to 1
RSP := newRSP;                 // load all 64 bits regardless of new mode
CS := newCS;                   // selector and descriptor
SS := newSS;                   // selector and descriptor
CPL := 3;
// swap GS.base and IA32_KERNEL_GS_BASE
tempGSB := GS.base;
GS.base := IA32_KERNEL_GS_BASE;
IA32_KERNEL_GS_BASE := tempGSB;
IF CR4.CET = 1 AND IA32_U_CET.SH_STK_EN = 1
    THEN SSP := IA32_PL3_SSP;
FI;
// update event-related state
IF NMI_unblock = 1
    THEN unblock NMIs;
FI;
IF pend_DB = 1 AND RFLAGS.TF = 1
    THEN pend a single-step debug exception (#DB) to be delivered after ERETU;
FI;

```

Flags Affected

All defined flags and fields in the RFLAGS register are potentially modified except for the VM flag.

Protected Mode Exceptions

#UD The ERETU instruction is not recognized in protected mode.

Real-Address Mode Exceptions

#UD The ERETU instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The ERETU instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD The ERETU instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD If CR4.FRED = 0.
If CPL > 0.
If the LOCK prefix is used.

#GP(0) If CSL > 0.
If the return is to 64-bit mode and the RIP image on the stack is not paging canonical.
If the return is to compatibility mode and the low 32 bits of the RIP image on the stack is not within the CS.limit being established.
If the RFLAGS image on the stack sets reserved bits or the VM bit (or fails to set bit 1).
If bits 13:12 of the RFLAGS image on the stack (corresponding to IOPL) are not zero.
If the value of bits 1:0 of the CS image on the stack (the target CPL) is not 3.
If the value of bits 63:16 of the CS image on the stack is not zero.
If the value of bits 1:0 of the SS image on the stack (the target CPL) is not 3.
If the SS image on the stack sets undefined bits (bits 31:19).
If the return is to compatibility mode and would load SSP with a value that sets any of bits 63:32.
If ERETU is loading CS and SS from descriptor tables (see Operation section) and either of the following apply:

- If the value of bits 15:2 of the CS image on the stack is 0.
- If the value of bits 15:2 of the SS image on the stack is 0.

#GP(selector) If ERETU is loading CS and SS from descriptor tables (see Operation section) and any of the following apply:

- A segment selector index is outside its descriptor table limits.
- A segment descriptor memory address is not paging canonical.
- The segment descriptor for CS does not indicate that it is a code segment.
- The segment descriptor for CS has both the D-bit and L-bit set.
- The segment descriptor for CS is non-conforming and its DPL is not 3.
- The segment descriptor for SS does not indicate that it is a writable data segment.
- The DPL of the segment descriptor for SS is not 3.

#SS(0) If an ordinary stack access would use an address that is not paging canonical or would cause a LASS violation.

#CP(FAR-RET/IRET) If the value in SSP differs from that in IA32_FRED_SSP0.

INCSSPD/INCSSPQ—Increment Shadow Stack Pointer

Opcode/ Instruction	Op / En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 OF AE /5 INCSSPD r32	R	V/V	CET_SS	Increment SSP by 4 * r32[7:0].
F3 REX.W OF AE /5 INCSSPQ r64	R	V/N.E.	CET_SS	Increment SSP by 8 * r64[7:0].

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
R	N/A	ModRM:r/m (r)	N/A	N/A	N/A

Description

This instruction can be used to increment the current shadow stack pointer by the operand size of the instruction times the unsigned 8-bit value specified by bits 7:0 in the source operand. The instruction performs a pop and discard of the first and last element on the shadow stack in the range specified by the unsigned 8-bit value in bits 7:0 of the source operand.

Operation

```

IF CPL = 3
    IF (CR4.CET & IA32_U_CET.SH_STK_EN) = 0
        THEN #UD; FI;
ELSE
    IF (CR4.CET & IA32_S_CET.SH_STK_EN) = 0
        THEN #UD; FI;
FI;

IF (operand size is 64-bit)
    THEN
        Range := R64[7:0];
        shadow_stack_load 8 bytes from SSP;
        IF Range > 0
            THEN shadow_stack_load 8 bytes from SSP + 8 * (Range - 1);
        FI;
        SSP := SSP + Range * 8;
    ELSE
        Range := R32[7:0];
        shadow_stack_load 4 bytes from SSP;
        IF Range > 0
            THEN shadow_stack_load 4 bytes from SSP + 4 * (Range - 1);
        FI;
        SSP := SSP + Range * 4;
FI;

```

Flags Affected

None.

Intel C/C++ Compiler Intrinsic Equivalent

```

INCSSPD void _incsspd(int);
INCSSPQ void _incsspq(int);

```

Protected Mode Exceptions

#UD If the LOCK prefix is used.
 If CR4.CET = 0.
 IF CPL = 3 and IA32_U_CET.SH_STK_EN = 0.
 IF CPL < 3 and IA32_S_CET.SH_STK_EN = 0.

#PF(fault-code) If a page fault occurs.

Real-Address Mode Exceptions

#UD The INCSSP instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The INCSSP instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

INT *n*/INT0/INT3/INT1—Call to Interrupt Procedure

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
CC	INT3	Z0	Valid	Valid	Generate breakpoint trap.
CD <i>ib</i>	INT <i>imm8</i>	I	Valid	Valid	Generate software interrupt with vector specified by immediate byte.
CE	INT0	Z0	Invalid	Valid	Generate overflow trap if overflow flag is 1.
F1	INT1	Z0	Valid	Valid	Generate debug trap.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A
I	<i>imm8</i>	N/A	N/A	N/A

Description

The INT *n* instruction generates a call to the interrupt or exception handler specified with the destination operand (see the section titled “Interrupts and Exceptions” in Chapter 6 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). The destination operand specifies a vector from 0 to 255, encoded as an 8-bit unsigned intermediate value. Each vector provides an index to a gate descriptor in the IDT. The first 32 vectors are reserved by Intel for system use. Some of these vectors are used for internally generated exceptions.

When IDT event delivery is used, the vector provides an index to a gate descriptor in the IDT. The first 32 vectors are reserved by Intel for system use. Some of these vectors are used for internally generated exceptions. When FRED event delivery is used, the vector is saved on the stack of the event handler.

The INT *n* instruction is the general mnemonic for executing a software-generated call to an interrupt handler. The INTO instruction is a special mnemonic for calling overflow exception (#OF), exception 4. The overflow interrupt checks the OF flag in the EFLAGS register and calls the overflow interrupt handler if the OF flag is set to 1. (The INTO instruction cannot be used in 64-bit mode.)

The INT3 instruction uses a one-byte opcode (CC) and is intended for calling the debug exception handler with a breakpoint exception (#BP). (This one-byte form is useful because it can replace the first byte of any instruction at which a breakpoint is desired, including other one-byte instructions, without overwriting other instructions.)

The INT1 instruction also uses a one-byte opcode (F1) and generates a debug exception (#DB) without setting any bits in DR6.¹ Hardware vendors may use the INT1 instruction for hardware debug. For that reason, Intel recommends software vendors instead use the INT3 instruction for software breakpoints.

An interrupt generated by the INTO, INT3, or INT1 instruction differs from one generated by INT *n* in the following ways:

- The normal IOPL checks do not occur in virtual-8086 mode. The interrupt is taken (without fault) with any IOPL value.
- The interrupt redirection enabled by the virtual-8086 mode extensions (VME) does not occur. The interrupt is always handled by a protected-mode handler.
- FRED event delivery uses event type 4 (software interrupt) for INT *n* but uses event type 5 (privileged software exception) for INT1 and event type 6 (software exception) for INT3 and INTO.

(These features do not pertain to CD01, CD03, or CD04, the “normal” 2-byte opcodes for INT 1, INT 3, and INT 4, respectively. Intel and Microsoft assemblers will not generate the CD03 opcode from any mnemonic, but this opcode can be created by direct numeric code definition or by self-modifying code.)

The operation and use of these instructions depends significantly on whether FRED transitions have been enabled by setting CR4.FRED. If CR4.FRED = 0, IDT event delivery is used; otherwise, FRED event delivery is used. The following sections apply as indicated.

1. The mnemonic ICEBP has also been used for the instruction with opcode F1.

With IDT Event Delivery

With IDT event delivery, the action of the INT *n* instruction (including the INTO, INT3, and INT1 instructions) is similar to that of a far call made with the CALL instruction. The primary difference is that with the INT *n* instruction, the EFLAGS register is pushed onto the stack before the return address. (The return address is a far address consisting of the current values of the CS and EIP registers.) Returns from interrupt procedures are handled with the IRET instruction, which pops the EFLAGS information and return address from the stack.

Each of the INT *n*, INTO, and INT3 instructions generates a general-protection exception (#GP) if the CPL is greater than the DPL value in the selected gate descriptor in the IDT. In contrast, the INT1 instruction can deliver a #DB even if the CPL is greater than the DPL of descriptor 1 in the IDT. (This behavior supports the use of INT1 by hardware vendors performing hardware debug.)

The vector specifies an interrupt descriptor in the interrupt descriptor table (IDT); that is, it provides index into the IDT. The selected interrupt descriptor in turn contains a pointer to an interrupt or exception handler procedure. In protected mode, the IDT contains an array of 8-byte descriptors, each of which is an interrupt gate, trap gate, or task gate. In real-address mode, the IDT is an array of 4-byte far pointers (2-byte code segment selector and a 2-byte instruction pointer), each of which point directly to a procedure in the selected segment. (Note that in real-address mode, the IDT is called the **interrupt vector table**, and its pointers are called interrupt vectors.)

The following decision table indicates which action in the lower portion of the table is taken given the conditions in the upper portion of the table. Each Y in the lower section of the decision table represents a procedure defined in the "Operation" section for this instruction (except #GP).

Table 3-54. Decision Table

PE	0	1	1	1	1	1	1	1
VM	-	-	-	-	-	0	1	1
IOPL	-	-	-	-	-	-	<3	=3
DPL/CPL RELATIONSHIP	-	DPL < CPL	-	DPL > CPL	DPL = CPL or C	DPL < CPL & NC	-	-
INTERRUPT TYPE	-	S/W	-	-	-	-	-	-
GATE TYPE	-	-	Task	Trap or Interrupt	Trap or Interrupt	Trap or Interrupt	Trap or Interrupt	Trap or Interrupt
REAL-ADDRESS-MODE	Y							
PROTECTED-MODE		Y	Y	Y	Y	Y	Y	Y
TRAP-OR-INTERRUPT-GATE				Y	Y	Y	Y	Y
INTER-PRIVILEGE-LEVEL-INTERRUPT						Y		
INTRA-PRIVILEGE-LEVEL-INTERRUPT					Y			
INTERRUPT-FROM-VIRTUAL-8086-MODE								Y
TASK-GATE			Y					
#GP		Y		Y			Y	

NOTES:

- Don't Care.
- Y Yes, action taken.
- Blank Action not taken.
- S/W Applies to INT *n*, INT3, and INTO, but not to INT1.

When the processor is executing in virtual-8086 mode, the IOPL determines the action of the INT *n* instruction. If the IOPL is less than 3, the processor generates a #GP(selector) exception; if the IOPL is 3, the processor executes

a protected mode interrupt to privilege level 0. The interrupt gate's DPL must be set to 3 and the target CPL of the interrupt handler procedure must be 0 to execute the protected mode interrupt to privilege level 0.

The interrupt descriptor table register (IDTR) specifies the base linear address and limit of the IDT. The initial base address value of the IDTR after the processor is powered up or reset is 0.

Refer to Chapter 6, "Procedure Calls, Interrupts, and Exceptions" and Chapter 18, "Control-flow Enforcement Technology (CET)" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for CET details.

With FRED Event Delivery

When FRED transitions are enabled, all of these events are delivered using FRED event delivery. Chapter 8, "Flexible Return and Event Delivery (FRED)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3 provides details. Like IDT event delivery, FRED event delivery is similar to a far call made with the CALL instruction, but the FRED stack frame is different. Returns from FRED event delivery use either the ERETS instruction or the ERETU instruction, depending on the CPL at the time of the original event.

FRED transitions can be enabled only in IA-32e mode. They cannot be enabled in real-address mode, protected mode, or virtual-8086 mode. FRED event delivery does not use the IDT, and its operation thus does not depend on any descriptor or DPL value or the value of IOPL.

Instruction ordering. Instructions following an INT *n* may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the INT *n* have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible). This applies also to the INTO, INT3, and INT1 instructions, but not to executions of INTO when EFLAGS.OF = 0.

Operation

The following operational description applies not only to the INT *n*, INTO, INT3, or INT1 instructions, but also to the delivery of external interrupts, nonmaskable interrupts (NMIs), and exceptions. Some of these events push onto the stack an error code.

The operational description specifies numerous checks whose failure may result in delivery of a nested exception. In these cases, the original event is not delivered.

The operational description specifies the error code delivered by any nested exception. In some cases, the error code is specified with a pseudofunction `error_code(num,idt,ext)`, where `idt` and `ext` are bit values. The pseudofunction produces an error code as follows: (1) if `idt` is 0, the error code is `(num & FCH) | ext`; (2) if `idt` is 1, the error code is `(num << 3) | 2 | ext`.

In many cases, the pseudofunction `error_code` is invoked with a pseudovariable `EXT`. The value of `EXT` depends on the nature of the event whose delivery encountered a nested exception: if that event is a software interrupt (INT *n*, INT3, or INTO), `EXT` is 0; otherwise (including INT1), `EXT` is 1.

```
IF PE = 0
  THEN
    GOTO REAL-ADDRESS-MODE;
  ELSE (* PE = 1 *)
    IF (EFLAGS.VM = 1 AND CR4.VME = 0 AND IOPL < 3 AND INT n)
      THEN
        #GP(0); (* Bit 0 of error code is 0 because INT n *)
      ELSE
        IF (EFLAGS.VM = 1 AND CR4.VME = 1 AND INT n)
          THEN
            Consult bit n of the software interrupt redirection bit map in the TSS;
            IF bit n is clear
              THEN (* redirect interrupt to 8086 program interrupt handler *)
                Push EFLAGS[15:0]; (* if IOPL < 3, save VIF in IF position and save IOPL position as 3 *)
                Push CS;
                Push IP;
                IF IOPL = 3
```

```

        THEN IF := 0; (* Clear interrupt flag *)
        ELSE VIF := 0; (* Clear virtual interrupt flag *)
    FI;
    TF := 0; (* Clear trap flag *)
    load CS and EIP (lower 16 bits only) from entry n in interrupt vector table referenced from TSS;
ELSE
    IF IOPL = 3
        THEN GOTO PROTECTED-MODE;
        ELSE #GP(0); (* Bit 0 of error code is 0 because INT n *)
    FI;
FI;
ELSE (* Protected mode, IA-32e mode, or virtual-8086 mode interrupt *)
    IF (IA32_EFER.LMA = 0)
        THEN (* Protected mode, or virtual-8086 mode interrupt *)
            GOTO PROTECTED-MODE;
        ELSE (* IA-32e mode interrupt *)
            GOTO IA-32e-MODE;
        FI;
    FI;
FI;

FI;

FI;
REAL-ADDRESS-MODE:
    IF ((vector_number << 2) + 3) is not within IDT limit
        THEN #GP; FI;
    IF stack not large enough for a 6-byte return information
        THEN #SS; FI;
    Push (EFLAGS[15:0]);
    IF := 0; (* Clear interrupt flag *)
    TF := 0; (* Clear trap flag *)
    AC := 0; (* Clear AC flag *)
    Push(CS);
    Push(IP);
    (* No error codes are pushed in real-address mode*)
    CS := IDT(Descriptor (vector_number << 2), selector);
    EIP := IDT(Descriptor (vector_number << 2), offset); (* 16 bit offset AND 0000FFFFH *)
END;

PROTECTED-MODE:
    IF ((vector_number << 3) + 7) is not within IDT limits
    or selected IDT descriptor is not an interrupt-, trap-, or task-gate type
        THEN #GP(error_code(vector_number,1,EXT)); FI;
        (* idt operand to error_code set because vector is used *)
    IF software interrupt (* Generated by INT n, INT3, or INTO; does not apply to INT1 *)
        THEN
            IF gate DPL < CPL (* PE = 1, DPL < CPL, software interrupt *)
                THEN #GP(error_code(vector_number,1,0)); FI;
                (* idt operand to error_code set because vector is used *)
                (* ext operand to error_code is 0 because INT n, INT3, or INTO*)
            FI;
        IF gate not present
            THEN #NP(error_code(vector_number,1,EXT)); FI;
            (* idt operand to error_code set because vector is used *)
        IF task gate (* Specified in the selected interrupt table descriptor *)
            THEN GOTO TASK-GATE;

```

```

        ELSE GOTO TRAP-OR-INTERRUPT-GATE; (* PE = 1, trap/interrupt gate *)
    FI;
END;

IA-32e-MODE:
    IF INTO and CS.L = 1 (64-bit mode)
        THEN #UD;
    FI;
    IF CR4.FRED = 0
        THEN
            IF ((vector_number << 4) + 15) is not in IDT limits
                or selected IDT descriptor is not an interrupt-, or trap-gate type
                THEN #GP(error_code(vector_number,1,EXT));
                (* idt operand to error_code set because vector is used *)
            FI;
            IF software interrupt (* Generated by INT n, INT3, or INTO; does not apply to INT1 *)
                THEN
                    IF gate DPL < CPL (* PE = 1, DPL < CPL, software interrupt *)
                        THEN #GP(error_code(vector_number,1,0));
                        (* idt operand to error_code set because vector is used *)
                        (* ext operand to error_code is 0 because INT n, INT3, or INTO*)
                    FI;
                    IF gate not present
                        THEN #NP(error_code(vector_number,1,EXT));
                        (* idt operand to error_code set because vector is used *)
                    FI;
                    GOTO TRAP-OR-INTERRUPT-GATE; (* Trap/interrupt gate *)
                ELSE (* CR4.FRED = 1 *)
                    FRED event delivery of software interrupt, exception, hardware interrupt, or non-maskable interrupt;
                END;
            END;

TASK-GATE: (* PE = 1, task gate *)
    Read TSS selector in task gate (IDT descriptor);
    IF local/global bit is set to local or index not within GDT limits
        THEN #GP(error_code(TSS selector,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
    Access TSS descriptor in GDT;
    IF TSS descriptor specifies that the TSS is busy (low-order 5 bits set to 00001)
        THEN #GP(error_code(TSS selector,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
    IF TSS not present
        THEN #NP(error_code(TSS selector,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
    SWITCH-TASKS (with nesting) to TSS;
    IF interrupt caused by fault with error code
        THEN
            IF stack limit does not allow push of error code
                THEN #SS(EXT); FI;
            Push(error code);
        FI;
        IF EIP not within code segment limit
            THEN #GP(EXT); FI;
    END;

```

TRAP-OR-INTERRUPT-GATE:

```
Read new code-segment selector for trap or interrupt gate (IDT descriptor);
IF new code-segment selector is NULL
    THEN #GP(EXT); FI; (* Error code contains NULL selector *)
IF new code-segment selector is not within its descriptor table limits
    THEN #GP(error_code(new code-segment selector,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
Read descriptor referenced by new code-segment selector;
IF descriptor does not indicate a code segment or new code-segment DPL > CPL
    THEN #GP(error_code(new code-segment selector,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
IF new code-segment descriptor is not present,
    THEN #NP(error_code(new code-segment selector,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
IF new code segment is non-conforming with DPL < CPL
    THEN
        IF VM = 0
            THEN
                GOTO INTER-PRIVILEGE-LEVEL-INTERRUPT;
                (* PE = 1, VM = 0, interrupt or trap gate, nonconforming code segment,
                DPL < CPL *)
            ELSE (* VM = 1 *)
                IF new code-segment DPL ≠ 0
                    THEN #GP(error_code(new code-segment selector,0,EXT));
                    (* idt operand to error_code is 0 because selector is used *)
                    GOTO INTERRUPT-FROM-VIRTUAL-8086-MODE; FI;
                    (* PE = 1, interrupt or trap gate, DPL < CPL, VM = 1 *)
                FI;
            ELSE (* PE = 1, interrupt or trap gate, DPL ≥ CPL *)
                IF VM = 1
                    THEN #GP(error_code(new code-segment selector,0,EXT));
                    (* idt operand to error_code is 0 because selector is used *)
                IF new code segment is conforming or new code-segment DPL = CPL
                    THEN
                        GOTO INTRA-PRIVILEGE-LEVEL-INTERRUPT;
                    ELSE (* PE = 1, interrupt or trap gate, nonconforming code segment, DPL > CPL *)
                        #GP(error_code(new code-segment selector,0,EXT));
                        (* idt operand to error_code is 0 because selector is used *)
                    FI;
                FI;
            END;
        FI;
    END;
```

INTER-PRIVILEGE-LEVEL-INTERRUPT:

```
(* PE = 1, interrupt or trap gate, non-conforming code segment, DPL < CPL *)
IF (IA32_EFER.LMA = 0) (* Not IA-32e mode *)
    THEN
        (* Identify stack-segment selector for new privilege level in current TSS *)
        IF current TSS is 32-bit
            THEN
                TSSstackAddress := (new code-segment DPL << 3) + 4;
                IF (TSSstackAddress + 5) > current TSS limit
                    THEN #TS(error_code(current TSS selector,0,EXT)); FI;
                    (* idt operand to error_code is 0 because selector is used *)
```

```

        NewSS := 2 bytes loaded from (TSS base + TSSstackAddress + 4);
        NewESP := 4 bytes loaded from (TSS base + TSSstackAddress);
    ELSE    (* current TSS is 16-bit *)
        TSSstackAddress := (new code-segment DPL << 2) + 2
        IF (TSSstackAddress + 3) > current TSS limit
            THEN #TS(error_code(current TSS selector,0,EXT)); FI;
            (* idt operand to error_code is 0 because selector is used *)
        NewSS := 2 bytes loaded from (TSS base + TSSstackAddress + 2);
        NewESP := 2 bytes loaded from (TSS base + TSSstackAddress);
FI;
IF NewSS is NULL
    THEN #TS(EXT); FI;
IF NewSS index is not within its descriptor-table limits
or NewSS RPL ≠ new code-segment DPL
    THEN #TS(error_code(NewSS,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
Read new stack-segment descriptor for NewSS in GDT or LDT;
IF new stack-segment DPL ≠ new code-segment DPL
or new stack-segment Type does not indicate writable data segment
    THEN #TS(error_code(NewSS,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
IF NewSS is not present
    THEN #SS(error_code(NewSS,0,EXT)); FI;
    (* idt operand to error_code is 0 because selector is used *)
    NewSSP := IA32_PLi_SSP (* where i = new code-segment DPL *)
ELSE (* IA-32e mode *)
    IF IDT-gate IST = 0
        THEN TSSstackAddress := (new code-segment DPL << 3) + 4;
        ELSE TSSstackAddress := (IDT gate IST << 3) + 28;
    FI;
    IF (TSSstackAddress + 7) > current TSS limit
        THEN #TS(error_code(current TSS selector,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
    NewRSP := 8 bytes loaded from (current TSS base + TSSstackAddress);
    NewSS := new code-segment DPL; (* NULL selector with RPL = new CPL *)
    IF IDT-gate IST = 0
        THEN
            NewSSP := IA32_PLi_SSP (* where i = new code-segment DPL *)
        ELSE
            NewSSPAddress = IA32_INTERRUPT_SSP_TABLE_ADDR + (IDT-gate IST << 3)
            (* Check if shadow stacks are enabled at CPL 0 *)
            IF ShadowStackEnabled(CPL 0)
                THEN NewSSP := 8 bytes loaded from NewSSPAddress; FI;
        FI;
    FI;
IF IDT gate is 32-bit
    THEN
        IF new stack does not have room for 24 bytes (error code pushed)
        or 20 bytes (no error code pushed)
            THEN #SS(error_code(NewSS,0,EXT)); FI;
            (* idt operand to error_code is 0 because selector is used *)
        FI
    ELSE
        IF IDT gate is 16-bit

```

```

        THEN
            IF new stack does not have room for 12 bytes (error code pushed)
            or 10 bytes (no error code pushed);
                THEN #SS(error_code(NewSS,0,EXT)); FI;
                (* idt operand to error_code is 0 because selector is used *)
        ELSE (* 64-bit IDT gate*)
            IF StackAddress is non-canonical
                THEN #SS(EXT); FI; (* Error code contains NULL selector *)
        FI;
    FI;
IF (IA32_EFER.LMA = 0) (* Not IA-32e mode *)
    THEN
        IF instruction pointer from IDT gate is not within new code-segment limits
            THEN #GP(EXT); FI; (* Error code contains NULL selector *)
        ESP := NewESP;
        SS := NewSS; (* Segment descriptor information also loaded *)
    ELSE (* IA-32e mode *)
        IF instruction pointer from IDT gate contains a non-canonical address
            THEN #GP(EXT); FI; (* Error code contains NULL selector *)
        RSP := NewRSP & FFFFFFFF0H;
        SS := NewSS;
    FI;
IF IDT gate is 32-bit
    THEN
        CS:EIP := Gate(CS:EIP); (* Segment descriptor information also loaded *)
    ELSE
        IF IDT gate 16-bit
            THEN
                CS:IP := Gate(CS:IP);
                (* Segment descriptor information also loaded *)
            ELSE (* 64-bit IDT gate *)
                CS:RIP := Gate(CS:RIP);
                (* Segment descriptor information also loaded *)
            FI;
    FI;
IF IDT gate is 32-bit
    THEN
        Push(far pointer to old stack);
        (* Old SS and ESP, 3 words padded to 4 *)
        Push(EFLAGS);
        Push(far pointer to return instruction);
        (* Old CS and EIP, 3 words padded to 4 *)
        Push(ErrorCode); (* If needed, 4 bytes *)
    ELSE
        IF IDT gate 16-bit
            THEN
                Push(far pointer to old stack);
                (* Old SS and SP, 2 words *)
                Push(EFLAGS(15:0));
                Push(far pointer to return instruction);
                (* Old CS and IP, 2 words *)
                Push(ErrorCode); (* If needed, 2 bytes *)
            ELSE (* 64-bit IDT gate *)
                Push(far pointer to old stack);

```

```

        (* Old SS and SP, each an 8-byte push *)
        Push(RFLAGS); (* 8-byte push *)
        Push(far pointer to return instruction);
        (* Old CS and RIP, each an 8-byte push *)
        Push(ErrorCode); (* If needed, 8-bytes *)

    FI;

FI;
IF ShadowStackEnabled(CPL) AND CPL = 3
    THEN
        IF IA32_EFER.LMA = 0
            THEN IA32_PL3_SSP := SSP;
            ELSE (* adjust so bits 63:N get the value of bit N-1, where N is the CPU's maximum linear-address width *)
                IA32_PL3_SSP := LA_adjust(SSP);

        FI;

FI;
CPL := new code-segment DPL;
CS(RPL) := CPL;
IF ShadowStackEnabled(CPL)
    oldSSP := SSP
    SSP := NewSSP
    IF SSP & 0x07 != 0
        THEN #GP(0); FI;
    (* Token and CS:LIP:oldSSP pushed on shadow stack must be contained in a naturally aligned 32-byte region *)
    IF (SSP & ~0x1F) != ((SSP - 24) & ~0x1F)
        #GP(0); FI;
    IF ((IA32_EFER.LMA and CS.L) = 0 AND SSP[63:32] != 0)
        THEN #GP(0); FI;
    expected_token_value = SSP (* busy bit - bit position 0 - must be clear *)
    new_token_value = SSP | BUSY_BIT (* Set the busy bit *)
    IF shadow_stack_lock_cmpxchg8b(SSP, new_token_value, expected_token_value) != expected_token_value
        THEN #GP(0); FI;
    IF oldSS.DPL != 3
        ShadowStackPush8B(oldCS); (* Padded with 48 high-order bits of 0 *)
        ShadowStackPush8B(oldCSBASE + oldRIP); (* Padded with 32 high-order bits of 0 for 32 bit LIP*)
        ShadowStackPush8B(oldSSP);

    FI;

FI;
IF EndbranchEnabled (CPL)
    IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH;
    IA32_S_CET.SUPPRESS = 0

FI;
IF IDT gate is interrupt gate
    THEN IF := 0 (* Interrupt flag set to 0, interrupts disabled *); FI;
TF := 0;
VM := 0;
RF := 0;
NT := 0;
END;

INTERRUPT-FROM-VIRTUAL-8086-MODE:
(* Identify stack-segment selector for privilege level 0 in current TSS *)
IF current TSS is 32-bit
    THEN
        IF TSS limit < 9

```



```

        THEN #TS(error_code(current TSS selector,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
        NewSS := 2 bytes loaded from (current TSS base + 8);
        NewESP := 4 bytes loaded from (current TSS base + 4);
    ELSE (* current TSS is 16-bit *)
        IF TSS limit < 5
            THEN #TS(error_code(current TSS selector,0,EXT)); FI;
            (* idt operand to error_code is 0 because selector is used *)
            NewSS := 2 bytes loaded from (current TSS base + 4);
            NewESP := 2 bytes loaded from (current TSS base + 2);
        FI;
    IF NewSS is NULL
        THEN #TS(EXT); FI; (* Error code contains NULL selector *)
    IF NewSS index is not within its descriptor table limits
    or NewSS RPL ≠ 0
        THEN #TS(error_code(NewSS,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
    Read new stack-segment descriptor for NewSS in GDT or LDT;
    IF new stack-segment DPL ≠ 0 or stack segment does not indicate writable data segment
        THEN #TS(error_code(NewSS,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
    IF new stack segment not present
        THEN #SS(error_code(NewSS,0,EXT)); FI;
        (* idt operand to error_code is 0 because selector is used *)
    NewSSP := IA32_PLO_SSP (* the new code-segment DPL must be 0 *)
    IF IDT gate is 32-bit
        THEN
            IF new stack does not have room for 40 bytes (error code pushed)
            or 36 bytes (no error code pushed)
                THEN #SS(error_code(NewSS,0,EXT)); FI;
                (* idt operand to error_code is 0 because selector is used *)
            ELSE (* IDT gate is 16-bit)
                IF new stack does not have room for 20 bytes (error code pushed)
                or 18 bytes (no error code pushed)
                    THEN #SS(error_code(NewSS,0,EXT)); FI;
                    (* idt operand to error_code is 0 because selector is used *)
            FI;
    IF instruction pointer from IDT gate is not within new code-segment limits
        THEN #GP(EXT); FI; (* Error code contains NULL selector *)
    tempEFLAGS := EFLAGS;
    VM := 0;
    TF := 0;
    RF := 0;
    NT := 0;
    IF service through interrupt gate
        THEN IF = 0; FI;
    TempSS := SS;
    TempESP := ESP;
    SS := NewSS;
    ESP := NewESP;
    (* Following pushes are 16 bits for 16-bit IDT gates and 32 bits for 32-bit IDT gates;
    Segment selector pushes in 32-bit mode are padded to two words *)
    Push(GS);
    Push(FS);

```

```

Push(DS);
Push(ES);
Push(TempSS);
Push(TempESP);
Push(TempEFlags);
Push(CS);
Push(EIP);
GS := 0; (* Segment registers made NULL, invalid for use in protected mode *)
FS := 0;
DS := 0;
ES := 0;
CS := Gate(CS); (* Segment descriptor information also loaded *)
CS(RPL) := 0;
CPL := 0;
IF IDT gate is 32-bit
    THEN
        EIP := Gate(instruction pointer);
    ELSE (* IDT gate is 16-bit *)
        EIP := Gate(instruction pointer) AND 0000FFFFH;
FI;
IF ShadowStackEnabled(0)
    oldSSP := SSP
    SSP := NewSSP
    IF SSP & 0x07 != 0
        THEN #GP(0); FI;
    (* Token and CS:LIP:oldSSP pushed on shadow stack must be contained in a naturally aligned 32-byte region *)
    IF (SSP & ~0x1F) != ((SSP - 24) & ~0x1F)
        #GP(0); FI;
IF ((IA32_EFER.LMA and CS.L) = 0 AND SSP[63:32] != 0)
    THEN #GP(0); FI;
expected_token_value = SSP (* busy bit - bit position 0 - must be clear *)
new_token_value = SSP | BUSY_BIT (* Set the busy bit *)
IF shadow_stack_lock_cmpxchg8b(SSP, new_token_value, expected_token_value) != expected_token_value
    THEN #GP(0); FI;
FI;
IF EndbranchEnabled (CPL)
    IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH;
    IA32_S_CET.SUPPRESS = 0
FI;
(* Start execution of new routine in Protected Mode *)
END;

```

INTRA-PRIVILEGE-LEVEL-INTERRUPT:

```

NewSSP = SSP;
CHECK_SS_TOKEN = 0
(* PE = 1, DPL = CPL or conforming segment *)
IF IA32_EFER.LMA = 1 (* IA-32e mode *)
    IF IDT-descriptor IST ≠ 0
        THEN
            TSSstackAddress := (IDT-descriptor IST << 3) + 28;
            IF (TSSstackAddress + 7) > TSS limit
                THEN #TS(error_code(current TSS selector,0,EXT)); FI;
            (* idt operand to error_code is 0 because selector is used *)
            NewRSP := 8 bytes loaded from (current TSS base + TSSstackAddress);

```

```

        ELSE NewRSP := RSP;
    FI;
    IF IDT-descriptor IST ≠ 0
        IF ShadowStackEnabled(CPL)
            THEN
                NewSSPAddress = IA32_INTERRUPT_SSP_TABLE_ADDR + (IDT gate IST << 3)
                NewSSP := 8 bytes loaded from NewSSPAddress
                CHECK_SS_TOKEN = 1
            FI;
        FI;
    FI;
    IF 32-bit gate (* implies IA32_EFER.LMA = 0 *)
        THEN
            IF current stack does not have room for 16 bytes (error code pushed)
                or 12 bytes (no error code pushed)
                THEN #SS(EXT); FI; (* Error code contains NULL selector *)
            ELSE IF 16-bit gate (* implies IA32_EFER.LMA = 0 *)
                IF current stack does not have room for 8 bytes (error code pushed)
                    or 6 bytes (no error code pushed)
                    THEN #SS(EXT); FI; (* Error code contains NULL selector *)
                ELSE (* IA32_EFER.LMA = 1, 64-bit gate*)
                    IF NewRSP contains a non-canonical address
                        THEN #SS(EXT); (* Error code contains NULL selector *)
                    FI;
                FI;
            FI;
        IF (IA32_EFER.LMA = 0) (* Not IA-32e mode *)
            THEN
                IF instruction pointer from IDT gate is not within new code-segment limit
                    THEN #GP(EXT); FI; (* Error code contains NULL selector *)
                ELSE
                    IF instruction pointer from IDT gate contains a non-canonical address
                        THEN #GP(EXT); FI; (* Error code contains NULL selector *)
                    RSP := NewRSP & FFFFFFFF0H;
                FI;
            IF IDT gate is 32-bit (* implies IA32_EFER.LMA = 0 *)
                THEN
                    Push (EFLAGS);
                    Push (far pointer to return instruction); (* 3 words padded to 4 *)
                    CS:EIP := Gate(CS:EIP); (* Segment descriptor information also loaded *)
                    Push (ErrorCode); (* If any *)
                ELSE
                    IF IDT gate is 16-bit (* implies IA32_EFER.LMA = 0 *)
                        THEN
                            Push (FLAGS);
                            Push (far pointer to return location); (* 2 words *)
                            CS:IP := Gate(CS:IP);
                            (* Segment descriptor information also loaded *)
                            Push (ErrorCode); (* If any *)
                        ELSE (* IA32_EFER.LMA = 1, 64-bit gate*)
                            Push(far pointer to old stack);
                            (* Old SS and SP, each an 8-byte push *)
                            Push(RFLAGS); (* 8-byte push *)
                            Push(far pointer to return instruction);
                            (* Old CS and RIP, each an 8-byte push *)
                        FI;
                    FI;
                FI;
            FI;
        FI;
    FI;

```

```

        Push(ErrorCode); (* If needed, 8 bytes *)
        CS:RIP := GATE(CS:RIP);
        (* Segment descriptor information also loaded *)
    FI;
FI;
CS(RPL) := CPL;
IF ShadowStackEnabled(CPL)
    IF CHECK_SS_TOKEN == 1
        THEN
            IF NewSSP & 0x07 != 0
                THEN #GP(0); FI;
            (* Token and CS:LIP:oldSSP pushed on shadow stack must be contained in a naturally aligned 32-byte region *)
            IF (NewSSP & ~0x1F) != ((NewSSP - 24) & ~0x1F)
                #GP(0); FI;

            IF ((IA32_EFER.LMA and CS.L) = 0 AND NewSSP[63:32] != 0)
                THEN #GP(0); FI;
            expected_token_value = NewSSP (* busy bit - bit position 0 - must be clear *)
            new_token_value = NewSSP | BUSY_BIT (* Set the busy bit *)
            IF shadow_stack_lock_cmpxchg8b(NewSSP, new_token_value, expected_token_value) != expected_token_value
                THEN #GP(0); FI;
        FI;
    (* Align to next 8 byte boundary *)
    tempSSP = SSP;
    Shadow_stack_store 4 bytes of 0 to (NewSSP - 4)
    SSP = newSSP & 0xFFFFFFFFFFFFF8H;
    (* push cs:lip:ssp on shadow stack *)
    ShadowStackPush8B(oldCS); (* Padded with 48 high-order bits of 0 *)
    ShadowStackPush8B(oldCSBASE + oldRIP); (* Padded with 32 high-order bits of 0 for 32 bit LIP*)
    ShadowStackPush8B(tempSSP);
FI;
IF EndbranchEnabled (CPL)
    IF CPL = 3
        THEN
            IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_U_CET.SUPPRESS = 0
        ELSE
            IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_S_CET.SUPPRESS = 0
    FI;
FI;
IF IDT gate is interrupt gate
    THEN IF := 0; FI; (* Interrupt flag set to 0; interrupts disabled *)
TF := 0;
NT := 0;
VM := 0;
RF := 0;
END;

```

Flags Affected

The EFLAGS register is pushed onto the stack. The IF, TF, NT, AC, RF, and VM flags may be cleared, depending on the mode of operation of the processor when the INT instruction is executed (see the "Operation" section). If the interrupt uses a task gate, any flags may be set or cleared, controlled by the EFLAGS image in the new task's TSS.

Protected Mode Exceptions

#GP(error_code)	If the instruction pointer in the IDT or in the interrupt, trap, or task gate is beyond the code segment limits. If the segment selector in the interrupt, trap, or task gate is NULL. If an interrupt, trap, or task gate, code segment, or TSS segment selector index is outside its descriptor table limits. If the vector selects a descriptor outside the IDT limits. If an IDT descriptor is not an interrupt, trap, or task gate. If an interrupt is generated by the INT n, INT3, or INTO instruction and the DPL of an interrupt, trap, or task gate is less than the CPL. If the segment selector in an interrupt or trap gate does not point to a segment descriptor for a code segment. If the segment selector for a TSS has its local/global bit set for local. If a TSS segment descriptor specifies that the TSS is busy or not available. If SSP in IA32_PLI_SSP (where i is the new CPL) is not 8 byte aligned. If the token and the stack frame to be pushed on shadow stack are not contained in a naturally aligned 32-byte region of the shadow stack. If “supervisor Shadow Stack” token on new shadow stack is marked busy. If destination mode is 32-bit or compatibility mode, but SSP address in “supervisor shadow stack” token is beyond 4GB. If SSP address in “supervisor shadow stack” token does not match SSP address in IA32_PLI_SSP (where i is the new CPL).
#SS(error_code)	If pushing the return address, flags, or error code onto the stack exceeds the bounds of the stack segment and no stack switch occurs. If the SS register is being loaded and the segment pointed to is marked not present. If pushing the return address, flags, error code, or stack segment pointer exceeds the bounds of the new stack segment when a stack switch occurs.
#NP(error_code)	If code segment, interrupt gate, trap gate, task gate, or TSS is not present.
#TS(error_code)	If the RPL of the stack segment selector in the TSS is not equal to the DPL of the code segment being accessed by the interrupt or trap gate. If DPL of the stack segment descriptor pointed to by the stack segment selector in the TSS is not equal to the DPL of the code segment descriptor for the interrupt or trap gate. If the stack segment selector in the TSS is NULL. If the stack segment for the TSS is not a writable data segment. If segment-selector index for stack segment is outside descriptor table limits.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
#AC(EXT)	If alignment checking is enabled, the gate DPL is 3, and a stack push is unaligned.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the interrupt vector number is outside the IDT limits.
#SS	If stack limit violation on push. If pushing the return address, flags, or error code onto the stack exceeds the bounds of the stack segment.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(error_code)	(For INT n, INTO, or BOUND instruction) If the IOPL is less than 3 or the DPL of the interrupt, trap, or task gate is not equal to 3.

If the instruction pointer in the IDT or in the interrupt, trap, or task gate is beyond the code segment limits.

If the segment selector in the interrupt, trap, or task gate is NULL.

If a interrupt gate, trap gate, task gate, code segment, or TSS segment selector index is outside its descriptor table limits.

If the vector selects a descriptor outside the IDT limits.

If an IDT descriptor is not an interrupt, trap, or task gate.

If an interrupt is generated by INT *n*, INT3, or INTO and the DPL of an interrupt, trap, or task gate is less than the CPL.

If the segment selector in an interrupt or trap gate does not point to a segment descriptor for a code segment.

If the segment selector for a TSS has its local/global bit set for local.

#SS(error_code) If the SS register is being loaded and the segment pointed to is marked not present.

If pushing the return address, flags, error code, stack segment pointer, or data segments exceeds the bounds of the stack segment.

#NP(error_code) If code segment, interrupt gate, trap gate, task gate, or TSS is not present.

#TS(error_code) If the RPL of the stack segment selector in the TSS is not equal to the DPL of the code segment being accessed by the interrupt or trap gate.

If DPL of the stack segment descriptor for the TSS's stack segment is not equal to the DPL of the code segment descriptor for the interrupt or trap gate.

If the stack segment selector in the TSS is NULL.

If the stack segment for the TSS is not a writable data segment.

If segment-selector index for stack segment is outside descriptor table limits.

#PF(fault-code) If a page fault occurs.

#OF If the INTO instruction is executed and the OF flag is set.

#UD If the LOCK prefix is used.

#AC(EXT) If alignment checking is enabled, the gate DPL is 3, and a stack push is unaligned.

Compatibility Mode Exceptions

#GP(error_code) If the instruction pointer in the 64-bit interrupt gate or trap gate is non-canonical.

If the segment selector in the 64-bit interrupt or trap gate is NULL.

If the vector selects a descriptor outside the IDT limits.

If the vector points to a gate which is in non-canonical space.

If the vector points to a descriptor which is not a 64-bit interrupt gate or a 64-bit trap gate.

If the descriptor pointed to by the gate selector is outside the descriptor table limit.

If the descriptor pointed to by the gate selector is in non-canonical space.

If the descriptor pointed to by the gate selector is not a code segment.

If the descriptor pointed to by the gate selector doesn't have the L-bit set, or has both the L-bit and D-bit set.

If the descriptor pointed to by the gate selector has DPL > CPL.

#GP(0) If SSP in IA32_PLi_SSP (where i is the new CPL) is not 8 byte aligned.

If the token and the stack frame to be pushed on shadow stack are not contained in a naturally aligned 32-byte region of the shadow stack.

If "supervisor shadow stack" token on new shadow stack is marked busy.

If destination mode is 32-bit or compatibility mode, but SSP address in "supervisor shadow stack" token is beyond 4GB.

If SSP address in "supervisor shadow stack" token does not match SSP address in IA32_PLi_SSP (where i is the new CPL).

If FRED event delivery would load RIP with a non-canonical address.

	If FRED event delivery would load SSP with an address that is not 8-byte aligned.
	If a shadow-stack access performed by FRED event delivery would use an address that is not canonical relative to the current paging mode or cause a LASS violation.
#SS(error_code)	If an ordinary stack access would use an address that is not canonical relative to the current paging mode or cause a LASS violation.
#NP(error_code)	If the 64-bit interrupt-gate, 64-bit trap-gate, or code segment is not present.
#TS(error_code)	If an attempt to load RSP from the TSS causes an access to non-canonical space.
	If the RSP from the TSS is outside descriptor table limits.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
#AC(EXT)	If alignment checking is enabled, the gate DPL is 3, and a stack push is unaligned.

64-Bit Mode Exceptions

Same exceptions as in [compatibility](#) mode.

IRET/IRETD/IRETQ—Interrupt Return

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
CF	IRET	Z0	Valid	Valid	Interrupt return (16-bit operand size).
CF	IRETD	Z0	Valid	Valid	Interrupt return (32-bit operand size).
REX.W + CF	IRETQ	Z0	Valid	N.E.	Interrupt return (64-bit operand size).

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

Returns program control from an exception or interrupt handler to a program or procedure that was interrupted by an exception, an external interrupt, or a software-generated interrupt. These instructions are also used to perform a return from a nested task. (A nested task is created when a CALL instruction is used to initiate a task switch or when an interrupt or exception causes a task switch to an interrupt or exception handler.) See the section titled “Task Linking” in Chapter 10 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

IRET and IRETD are mnemonics for the same opcode. The IRETD mnemonic (interrupt return double) is intended for use when returning from an interrupt when using the 32-bit operand size; however, most assemblers use the IRET mnemonic interchangeably for both operand sizes.

In Real-Address Mode, the IRET instruction performs a far return to the interrupted program or procedure. During this operation, the processor pops the return instruction pointer, return code segment selector, and EFLAGS image from the stack to the EIP, CS, and EFLAGS registers, respectively, and then resumes execution of the interrupted program or procedure.

In Protected Mode, the action of the IRET instruction depends on the settings of the NT (nested task) and VM flags in the EFLAGS register and the VM flag in the EFLAGS image stored on the current stack. Depending on the setting of these flags, the processor performs the following types of interrupt returns:

- Return from virtual-8086 mode.
- Return to virtual-8086 mode.
- Intra-privilege level return.
- Inter-privilege level return.
- Return from nested task (task switch).

If the NT flag (EFLAGS register) is cleared, the IRET instruction performs a far return from the interrupt procedure, without a task switch. The code segment being returned to must be equally or less privileged than the interrupt handler routine (as indicated by the RPL field of the code segment selector popped from the stack).

As with a real-address mode interrupt return, the IRET instruction pops the return instruction pointer, return code segment selector, and EFLAGS image from the stack to the EIP, CS, and EFLAGS registers, respectively, and then resumes execution of the interrupted program or procedure. If the return is to another privilege level, the IRET instruction also pops the stack pointer and SS from the stack, before resuming program execution. If the return is to virtual-8086 mode, the processor also pops the data segment registers from the stack.

If the NT flag is set, the IRET instruction performs a task switch (return) from a nested task (a task called with a CALL instruction, an interrupt, or an exception) back to the calling or interrupted task. The updated state of the task executing the IRET instruction is saved in its TSS. If the task is re-entered later, the code that follows the IRET instruction is executed.

If the NT flag is set and the processor is in IA-32e mode, the IRET instruction causes a general protection exception.

If nonmaskable interrupts (NMIs) are blocked (see Section 7.7.1, “Handling Multiple NMIs” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A), execution of the IRET instruction unblocks NMIs.

This unblocking occurs even if the instruction causes a fault. In such a case, NMIs are unmasked before the exception handler is invoked.

In 64-bit mode, the instruction's default operation size is 32 bits. Use of the REX.W prefix promotes operation to 64 bits (IRETQ). See the summary chart at the beginning of this section for encoding data and limits.

Refer to Chapter 6, "Procedure Calls, Interrupts, and Exceptions" and Chapter 18, "Control-flow Enforcement Technology (CET)" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for CET details.

When FRED transitions are enabled, an execution of IRET that would change CPL causes a general-protection exception, as does an execution of IRET that would enter compatibility mode when CPL is 0.

Instruction ordering. IRET is a serializing instruction. See Section 11.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

See "Changes to Instruction Behavior in VMX Non-Root Operation" in Chapter 28 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C, for more information about the behavior of this instruction in VMX non-root operation.

Operation

```
IF PE = 0
    THEN GOTO REAL-ADDRESS-MODE;
ELSIF (IA32_EFER.LMA = 0)
    THEN
        IF (EFLAGS.VM = 1)
            THEN GOTO RETURN-FROM-VIRTUAL-8086-MODE;
            ELSE GOTO PROTECTED-MODE;
        FI;
    ELSE GOTO IA-32e-MODE;
FI;

REAL-ADDRESS-MODE:
    IF OperandSize = 32
        THEN
            EIP := Pop();
            CS := Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            tempEFLAGS := Pop();
            EFLAGS := (tempEFLAGS AND 257FD5H) OR (EFLAGS AND 1A0000H);
        ELSE (* OperandSize = 16 *)
            EIP := Pop(); (* 16-bit pop; clear upper 16 bits *)
            CS := Pop(); (* 16-bit pop *)
            EFLAGS[15:0] := Pop();
        FI;
    END;

RETURN-FROM-VIRTUAL-8086-MODE:
(* Processor is in virtual-8086 mode when IRET is executed and stays in virtual-8086 mode *)
    IF IOPL = 3 (* Virtual mode: PE = 1, VM = 1, IOPL = 3 *)
        THEN IF OperandSize = 32
            THEN
                EIP := Pop();
                CS := Pop(); (* 32-bit pop, high-order 16 bits discarded *)
                EFLAGS := Pop();
                (* VM, IOPL, VIP and VIF EFLAG bits not modified by pop *)
                IF EIP not within CS limit
                    THEN #GP(0); FI;
            ELSE (* OperandSize = 16 *)
                EIP := Pop(); (* 16-bit pop; clear upper 16 bits *)
```

```

        CS := Pop(); (* 16-bit pop *)
        EFLAGS[15:0] := Pop(); (* IOPL in EFLAGS not modified by pop *)
        IF EIP not within CS limit
            THEN #GP(0); FI;
        FI;
    ELSE
        #GP(0); (* Trap to virtual-8086 monitor: PE = 1, VM = 1, IOPL < 3 *)
    FI;
END;

PROTECTED-MODE:
    IF NT = 1
        THEN GOTO TASK-RETURN; (* PE = 1, VM = 0, NT = 1 *)
    FI;
    IF OperandSize = 32
        THEN
            EIP := Pop();
            CS := Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            tempEFLAGS := Pop();
        ELSE (* OperandSize = 16 *)
            EIP := Pop(); (* 16-bit pop; clear upper bits *)
            CS := Pop(); (* 16-bit pop *)
            tempEFLAGS := Pop(); (* 16-bit pop; clear upper bits *)
        FI;
    IF tempEFLAGS(VM) = 1 and CPL = 0
        THEN GOTO RETURN-TO-VIRTUAL-8086-MODE;
    ELSE GOTO PROTECTED-MODE-RETURN;
    FI;

TASK-RETURN: (* PE = 1, VM = 0, NT = 1 *)
    SWITCH-TASKS (without nesting) to TSS specified in link field of current TSS;
    Mark the task just abandoned as NOT BUSY;
    IF EIP is not within CS limit
        THEN #GP(0); FI;
    END;

RETURN-TO-VIRTUAL-8086-MODE:
    (* Interrupted procedure was in virtual-8086 mode: PE = 1, CPL=0, VM = 1 in flag image *)
    (* If shadow stack or indirect branch tracking at CPL3 then #GP(0) *)
    IF CR4.CET AND (IA32_U_CET.ENDBR_EN OR IA32_U_CET.SHSTK_EN)
        THEN #GP(0); FI;
    shadowStackEnabled = ShadowStackEnabled(CPL)
    IF EIP not within CS limit
        THEN #GP(0); FI;
    EFLAGS := tempEFLAGS;
    ESP := Pop();
    SS := Pop(); (* Pop 2 words; throw away high-order word *)
    ES := Pop(); (* Pop 2 words; throw away high-order word *)
    DS := Pop(); (* Pop 2 words; throw away high-order word *)
    FS := Pop(); (* Pop 2 words; throw away high-order word *)
    GS := Pop(); (* Pop 2 words; throw away high-order word *)
    IF shadowStackEnabled
        (* check if 8 byte aligned *)
        IF SSP AND 0x7 != 0

```

```

        THEN #CP(FAR-RET/IRET); FI;
FI;

CPL := 3;
(* Resume execution in Virtual-8086 mode *)
tempOldSSP = SSP;
(* Now past all faulting points; safe to free the token. The token free is done using the old SSP
 * and using a supervisor override as old CPL was a supervisor privilege level *)
IF shadowStackEnabled
    expected_token_value = tempOldSSP | BUSY_BIT    (* busy bit - bit position 0 - must be set *)
    new_token_value = tempOldSSP                    (* clear the busy bit *)
    shadow_stack_lock_cmpxchg8b(tempOldSSP, new_token_value, expected_token_value)
FI;
END;

PROTECTED-MODE-RETURN: (* PE = 1 *)
    IF CS(RPL) > CPL
        THEN GOTO RETURN-TO-OUTER-PRIVILEGE-LEVEL;
        ELSE GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL; FI;
END;

RETURN-TO-OUTER-PRIVILEGE-LEVEL:
    IF OperandSize = 32
        THEN
            tempESP := Pop();
            tempSS := Pop(); (* 32-bit pop, high-order 16 bits discarded *)
        ELSE IF OperandSize = 16
            THEN
                tempESP := Pop(); (* 16-bit pop; clear upper bits *)
                tempSS := Pop(); (* 16-bit pop *)
            ELSE (* OperandSize = 64 *)
                tempRSP := Pop();
                tempSS := Pop(); (* 64-bit pop, high-order 48 bits discarded *)
            FI;
        IF new mode ≠ 64-Bit Mode
            THEN
                IF EIP is not within CS limit
                    THEN #GP(0); FI;
                ELSE (* new mode = 64-bit mode *)
                    IF RIP is non-canonical
                        THEN #GP(0); FI;
                    FI;
            EFLAGS(CF, PF, AF, ZF, SF, TF, DF, OF, NT) := tempEFLAGS;
            IF OperandSize = 32 or OperandSize = 64
                THEN EFLAGS(RF, AC, ID) := tempEFLAGS; FI;
            IF CPL ≤ IOPL
                THEN EFLAGS(IF) := tempEFLAGS; FI;
            IF CPL = 0
                THEN
                    EFLAGS(IOPL) := tempEFLAGS;
                    IF OperandSize = 32 or OperandSize = 64
                        THEN EFLAGS(VIF, VIP) := tempEFLAGS; FI;
            FI;
            IF ShadowStackEnabled(CPL)

```

```

(* check if 8 byte aligned *)
IF SSP AND 0x7 != 0
    THEN #CP(FAR-RET/IRET); FI;
IF CS(RPL) != 3
    THEN
        tempSsCS = shadow_stack_load 8 bytes from SSP+16;
        tempSsLIP = shadow_stack_load 8 bytes from SSP+8;
        tempSSP = shadow_stack_load 8 bytes from SSP;
        SSP = SSP + 24;
        (* Do 64 bit compare to detect bits beyond 15 being set *)
        tempCS = CS; (* zero padded to 64 bit *)
        IF tempCS != tempSsCS
            THEN #CP(FAR-RET/IRET); FI;
        (* Do 64 bit compare; pad CSBASE+RIP with 0 for 32 bit LIP *)
        IF CSBASE + RIP != tempSsEIP
            THEN #CP(FAR-RET/IRET); FI;
        (* check if 4 byte aligned *)
        IF tempSSP AND 0x3 != 0
            THEN #CP(FAR-RET/IRET); FI;

FI;
FI;
tempOldCPL = CPL;
CPL := CS(RPL);
IF OperandSize = 64
    THEN
        RSP := tempRSP;
        SS := tempSS;
    ELSE
        ESP := tempESP;
        SS := tempSS;
FI;
IF new mode != 64-Bit Mode
    THEN
        IF EIP is not within CS limit
            THEN #GP(0); FI;
        ELSE (* new mode = 64-bit mode *)
            IF RIP is non-canonical
                THEN #GP(0); FI;
FI;
tempOldSSP = SSP;
IF ShadowStackEnabled(CPL)
    IF CPL = 3
        THEN tempSSP := IA32_PL3_SSP; FI;
IF ((IA32_EFER.LMA AND CS.L) = 0 AND tempSSP[63:32] != 0) OR
   ((IA32_EFER.LMA AND CS.L) = 1 AND tempSSP is not canonical relative to the current paging mode)
    THEN #GP(0); FI;
SSP := tempSSP
FI;
(* Now past all faulting points; safe to free the token. The token free is done using the old SSP
 * and using a supervisor override as old CPL was a supervisor privilege level *)
IF ShadowStackEnabled(tempOldCPL)
    expected_token_value = tempOldSSP | BUSY_BIT (* busy bit - bit position 0 - must be set *)
    new_token_value = tempOldSSP (* clear the busy bit *)
    shadow_stack_lock_cmpxchg8b(tempOldSSP, new_token_value, expected_token_value)

```

```

FI;

FOR each SegReg in (ES, FS, GS, and DS)
DO
    tempDesc := descriptor cache for SegReg (* hidden part of segment register *)
    IF (SegmentSelector == NULL) OR (tempDesc(DPL) < CPL AND tempDesc(Type) is (data or non-conforming code)))
        THEN (* Segment register invalid *)
            SegmentSelector := 0; (*Segment selector becomes null*)
    FI;
OD;
END;

RETURN-TO-SAME-PRIVILEGE-LEVEL: (* PE = 1, RPL = CPL *)
    IF new mode ≠ 64-Bit Mode
        THEN
            IF EIP is not within CS limit
                THEN #GP(0); FI;
            ELSE (* new mode = 64-bit mode *)
                IF RIP is non-canonical
                    THEN #GP(0); FI;
            FI;
        EFLAGS (CF, PF, AF, ZF, SF, TF, DF, OF, NT) := tempEFLAGS;
        IF OperandSize = 32 or OperandSize = 64
            THEN EFLAGS(RF, AC, ID) := tempEFLAGS; FI;
        IF CPL ≤ IOPL
            THEN EFLAGS(IF) := tempEFLAGS; FI;
        IF CPL = 0
            THEN
                EFLAGS(IOPL) := tempEFLAGS;
                IF OperandSize = 32 or OperandSize = 64
                    THEN EFLAGS(VIF, VIP) := tempEFLAGS; FI;
            FI;
        IF ShadowStackEnabled(CPL)
            IF SSP AND 0x7 != 0 (* check if aligned to 8 bytes *)
                THEN #CP(FAR-RET/IRET); FI;
            tempSsCS = shadow_stack_load 8 bytes from SSP+16;
            tempSsLIP = shadow_stack_load 8 bytes from SSP+8;
            tempSSP = shadow_stack_load 8 bytes from SSP;
            SSP = SSP + 24;
            tempCS = CS; (* zero padded to 64 bit *)
            IF tempCS != tempSsCS (* 64 bit compare; CS zero padded to 64 bits *)
                THEN #CP(FAR-RET/IRET); FI;
            IF CSBASE + RIP != tempSsLIP (* 64 bit compare; CSBASE+RIP zero padded to 64 bit for 32 bit LIP *)
                THEN #CP(FAR-RET/IRET); FI;
            IF tempSSP AND 0x3 != 0 (* check if aligned to 4 bytes *)
                THEN #CP(FAR-RET/IRET); FI;
            IF ((IA32_EFER.LMA AND CS.L) = 0 AND tempSSP[63:32] != 0) OR
                ((IA32_EFER.LMA AND CS.L) = 1 AND tempSSP is not canonical relative to the current paging mode)
                THEN #GP(0); FI;
            FI;
        IF ShadowStackEnabled(CPL)
            IF IA32_EFER.LMA = 1
                (* In IA-32e-mode the IRET may be switching stacks if the interrupt/exception was delivered
                through an IDT with a non-zero IST *)

```

(* In IA-32e mode for same CPL IRET there is always a stack switch. The below check verifies if the stack switch was to self stack and if so, do not try to free the token on this shadow stack. If the tempSSP was not to same stack then there was a stack switch so do attempt to free the token *)

```

    IF tempSSP != SSP
        THEN
            expected_token_value = SSP | BUSY_BIT          (* busy bit - bit position 0 - must be set *)
            new_token_value = SSP                          (* clear the busy bit *)
            shadow_stack_lock_cmpxchg8b(SSP, new_token_value, expected_token_value)
        FI;
    FI;
    SSP := tempSSP
FI;
END;
```

IA-32e-MODE:

```

    IF NT = 1
        THEN #GP(0);
    ELSE IF OperandSize = 32
        THEN
            EIP := Pop();
            CS := Pop();
            tempEFLAGS := Pop();
        ELSE IF OperandSize = 16
            THEN
                EIP := Pop(); (* 16-bit pop; clear upper bits *)
                CS := Pop(); (* 16-bit pop *)
                tempEFLAGS := Pop(); (* 16-bit pop; clear upper bits *)
            FI;
        ELSE (* OperandSize = 64 *)
            THEN
                RIP := Pop();
                CS := Pop(); (* 64-bit pop, high-order 48 bits discarded *)
                tempRFLAGS := Pop();
```

```

FI;
IF CS.RPL < CPL or (CR4.FRED = 1 and CS.RPL > CPL)
    THEN #GP(CS.selector); FI;
```

```

IF CS.RPL > CPL
    THEN GOTO RETURN-TO-OUTER-PRIVILEGE-LEVEL;
ELSE (* CS.RPL = CPL *)
    IF CR4.FRED = 1 and CPL = 0 and CS.L = 0
        THEN #GP(CS.selector); FI;
    IF instruction began in 64-Bit Mode
        THEN
            IF OperandSize = 32
                THEN
                    ESP := Pop();
                    SS := Pop(); (* 32-bit pop, high-order 16 bits discarded *)
                ELSE IF OperandSize = 16
                    THEN
                        ESP := Pop(); (* 16-bit pop; clear upper bits *)
                        SS := Pop(); (* 16-bit pop *)
                    ELSE (* OperandSize = 64 *)
                        RSP := Pop();
                        SS := Pop(); (* 64-bit pop, high-order 48 bits discarded *)
```

```

        FI;
    FI;
    GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL; FI;
END;

```

Flags Affected

All the flags and fields in the EFLAGS register are potentially modified, depending on the mode of operation of the processor. If performing a return from a nested task to a previous task, the EFLAGS register will be modified according to the EFLAGS image stored in the previous task's TSS.

Protected Mode Exceptions

#GP(0)	If the return code or stack segment selector is NULL. If the return instruction pointer is not within the return code segment limit.
#GP(selector)	If a segment selector index is outside its descriptor table limits. If the return code segment selector RPL is less than the CPL. If the DPL of a conforming-code segment is greater than the return code segment selector RPL. If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment selector. If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector. If the stack segment is not a writable data segment. If the stack segment selector RPL is not equal to the RPL of the return code segment selector. If the segment descriptor for a code segment does not indicate it is a code segment. If the segment selector for a TSS has its local/global bit set for local. If a TSS segment descriptor specifies that the TSS is not busy. If a TSS segment descriptor specifies that the TSS is not available.
#SS(0)	If the top bytes of stack are not within stack limits. If the return stack segment is not present.
#NP (selector)	If the return code segment is not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference occurs when the CPL is 3 and alignment checking is enabled.
#UD	If the LOCK prefix is used.
#CP (Far-RET/IRET)	If the previous SSP from shadow stack (when returning to CPL <3) or from IA32_PL3_SSP (returning to CPL 3) is not 4 byte aligned. If returning to 32-bit or compatibility mode and the previous SSP from shadow stack (when returning to CPL <3) or from IA32_PL3_SSP (returning to CPL 3) is beyond 4GB. If return instruction pointer from stack and shadow stack do not match.

Real-Address Mode Exceptions

#GP	If the return instruction pointer is not within the return code segment limit.
#SS	If the top bytes of stack are not within stack limits.

Virtual-8086 Mode Exceptions

#GP(0)	If the return instruction pointer is not within the return code segment limit. If IOPL not equal to 3.
#PF(fault-code)	If a page fault occurs.
#SS(0)	If the top bytes of stack are not within stack limits.
#AC(0)	If an unaligned memory reference occurs and alignment checking is enabled.

#UD If the LOCK prefix is used.

Compatibility Mode Exceptions

#GP(0) If EFLAGS.NT[bit 14] = 1.

Other exceptions same as in Protected Mode.

64-Bit Mode Exceptions

#GP(0) If EFLAGS.NT[bit 14] = 1.

If the return code segment selector is NULL.

If the stack segment selector is NULL going back to compatibility mode.

If the stack segment selector is NULL going back to CPL3 64-bit mode.

If a NULL stack segment selector RPL is not equal to CPL going back to non-CPL3 64-bit mode.

If the return instruction pointer is not within the return code segment limit.

If the return instruction pointer is non-canonical.

#GP(Selector) If a segment selector index is outside its descriptor table limits.

If a segment descriptor memory address is non-canonical.

If the segment descriptor for a code segment does not indicate it is a code segment.

If the proposed new code segment descriptor has both the D-bit and L-bit set.

If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment selector.

If CPL is greater than the RPL of the code segment selector.

If FRED transitions are enabled and CPL is less than the RPL of the code segment selector.

If FRED transitions are enabled and CPL = 0 and IRET would enter compatibility mode.

If the DPL of a conforming-code segment is greater than the return code segment selector RPL.

If the stack segment is not a writable data segment.

If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector.

If the stack segment selector RPL is not equal to the RPL of the return code segment selector.

#SS(0) If an attempt to pop a value off the stack violates the SS limit.

If an attempt to pop a value off the stack causes a non-canonical address to be referenced.

If the return stack segment is not present.

#NP (selector) If the return code segment is not present.

#PF(fault-code) If a page fault occurs.

#AC(0) If an unaligned memory reference occurs when the CPL is 3 and alignment checking is enabled.

#UD If the LOCK prefix is used.

#CP (Far-RET/IRET) If the previous SSP from shadow stack (when returning to CPL <3) or from IA32_PL3_SSP (returning to CPL 3) is not 4 byte aligned.

If returning to 32-bit or compatibility mode and the previous SSP from shadow stack (when returning to CPL <3) or from IA32_PL3_SSP (returning to CPL 3) is beyond 4GB.

If return instruction pointer from stack and shadow stack do not match.

JMP—Jump

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
EB cb	JMP rel8	D	Valid	Valid	Jump short, RIP = RIP + 8-bit displacement sign extended to 64-bits.
E9 cw	JMP rel16	D	N.S.	Valid	Jump near, relative, displacement relative to next instruction. Not supported in 64-bit mode.
E9 cd	JMP rel32	D	Valid	Valid	Jump near, relative, RIP = RIP + 32-bit displacement sign extended to 64-bits.
FF /4	JMP r/m16	M	N.S.	Valid	Jump near, absolute indirect, address = zero-extended r/m16. Not supported in 64-bit mode.
FF /4	JMP r/m32	M	N.S.	Valid	Jump near, absolute indirect, address given in r/m32. Not supported in 64-bit mode.
FF /4	JMP r/m64	M	Valid	N.E.	Jump near, absolute indirect, RIP = 64-Bit offset from register or memory.
EA cd	JMP ptr16:16	S	Inv.	Valid	Jump far, absolute, address given in operand.
EA cp	JMP ptr16:32	S	Inv.	Valid	Jump far, absolute, address given in operand.
FF /5	JMP m16:16	M	Valid	Valid	Jump far, absolute indirect, address given in m16:16.
FF /5	JMP m16:32	M	Valid	Valid	Jump far, absolute indirect, address given in m16:32.
REX.W FF /5	JMP m16:64	M	Valid	N.E.	Jump far, absolute indirect, address given in m16:64.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
S	Segment + Absolute Address	N/A	N/A	N/A
D	Offset	N/A	N/A	N/A
M	ModRM:r/m (r)	N/A	N/A	N/A

Description

Transfers program control to a different point in the instruction stream without recording return information. The destination (target) operand specifies the address of the instruction being jumped to. This operand can be an immediate value, a general-purpose register, or a memory location.

This instruction can be used to execute four different types of jumps:

- Near jump—A jump to an instruction within the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intrasegment jump.
- Short jump—A near jump where the jump range is limited to –128 to +127 from the current EIP value.
- Far jump—A jump to an instruction located in a different segment than the current code segment but at the same privilege level, sometimes referred to as an intersegment jump.
- Task switch—A jump to an instruction located in a different task.

A task switch can only be executed in protected mode (see Chapter 10, in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, for information on performing task switches with the JMP instruction).

Near and Short Jumps. When executing a near jump, the processor jumps to the address (within the current code segment) that is specified with the target operand. The target operand specifies either an absolute offset (that is an offset from the base of the code segment) or a relative offset (a signed displacement relative to the current

value of the instruction pointer in the EIP register). A near jump to a relative offset of 8-bits (*rel8*) is referred to as a short jump. The CS register is not changed on near and short jumps.

An absolute offset is specified indirectly in a general-purpose register or a memory location (*r/m16* or *r/m32*). The operand-size attribute determines the size of the target operand (16 or 32 bits). Absolute offsets are loaded directly into the EIP register. If the operand-size attribute is 16, the upper two bytes of the EIP register are cleared, resulting in a maximum instruction pointer size of 16 bits.

A relative offset (*rel8*, *rel16*, or *rel32*) is generally specified as a label in assembly code, but at the machine code level, it is encoded as a signed 8-, 16-, or 32-bit immediate value. This value is added to the value in the EIP register. (Here, the EIP register contains the address of the instruction following the JMP instruction). When using relative offsets, the opcode (for short vs. near jumps) and the operand-size attribute (for near relative jumps) determines the size of the target operand (8, 16, or 32 bits).

Far Jumps in Real-Address or Virtual-8086 Mode. When executing a far jump in real-address or virtual-8086 mode, the processor jumps to the code segment and offset specified with the target operand. Here the target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). With the pointer method, the segment and address of the called procedure is encoded in the instruction, using a 4-byte (16-bit operand size) or 6-byte (32-bit operand size) far address immediate. With the indirect method, the target operand specifies a memory location that contains a 4-byte (16-bit operand size) or 6-byte (32-bit operand size) far address. The far address is loaded directly into the CS and EIP registers. If the operand-size attribute is 16, the upper two bytes of the EIP register are cleared.

Far Jumps in Protected Mode. When the processor is operating in protected mode, the JMP instruction can be used to perform the following three types of far jumps:

- A far jump to a conforming or non-conforming code segment.
- A far jump through a call gate.
- A task switch.

(The JMP instruction cannot be used to perform inter-privilege-level far jumps.)

In protected mode, the processor always uses the segment selector part of the far address to access the corresponding descriptor in the GDT or LDT. The descriptor type (code segment, call gate, task gate, or TSS) and access rights determine the type of jump to be performed.

If the selected descriptor is for a code segment, a far jump to a code segment at the same privilege level is performed. (If the selected code segment is at a different privilege level and the code segment is non-conforming, a general-protection exception is generated.) A far jump to the same privilege level in protected mode is very similar to one carried out in real-address or virtual-8086 mode. The target operand specifies an absolute far address either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*). The operand-size attribute determines the size of the offset (16 or 32 bits) in the far address. The new code segment selector and its descriptor are loaded into CS register, and the offset from the instruction is loaded into the EIP register. Note that a call gate (described in the next paragraph) can also be used to perform far call to a code segment at the same privilege level. Using this mechanism provides an extra level of indirection and is the preferred method of making jumps between 16-bit and 32-bit code segments.

When executing a far jump through a call gate, the segment selector specified by the target operand identifies the call gate. (The offset part of the target operand is ignored.) The processor then jumps to the code segment specified in the call gate descriptor and begins executing the instruction at the offset specified in the call gate. No stack switch occurs. Here again, the target operand can specify the far address of the call gate either directly with a pointer (*ptr16:16* or *ptr16:32*) or indirectly with a memory location (*m16:16* or *m16:32*).

Executing a task switch with the JMP instruction is somewhat similar to executing a jump through a call gate. Here the target operand specifies the segment selector of the task gate for the task being switched to (and the offset part of the target operand is ignored). The task gate in turn points to the TSS for the task, which contains the segment selectors for the task's code and stack segments. The TSS also contains the EIP value for the next instruction that was to be executed before the task was suspended. This instruction pointer value is loaded into the EIP register so that the task begins executing again at this next instruction.

The JMP instruction can also specify the segment selector of the TSS directly, which eliminates the indirection of the task gate. See Chapter 10 in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for detailed information on the mechanics of a task switch.

When execution of a JMP instruction effects a task switch, the nested task flag (NT) is not set in the EFLAGS register and the new TSS's previous task link field is not loaded with the old task's TSS selector. A return to the previous task can thus not be carried out by executing the IRET instruction. Switching tasks with the JMP instruction differs in this regard from the CALL instruction which does set the NT flag and save the previous task link information, allowing a return to the calling task with an IRET instruction.

Refer to Chapter 6, "Procedure Calls, Interrupts, and Exceptions" and Chapter 18, "Control-flow Enforcement Technology (CET)" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for CET details.

In 64-Bit Mode. The instruction's operation size is fixed at 64 bits. If a selector points to a gate, then RIP equals the 64-bit displacement taken from gate; else RIP equals the zero-extended offset from the far pointer referenced in the instruction.

When FRED transitions are enabled, an execution of far JMP that references a call gate causes a general-protection exception, as does an execution of far JMP that would enter compatibility mode when CPL is 0.

See the summary chart at the beginning of this section for encoding data and limits.

Instruction ordering. Instructions following a far jump may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the far jump have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Instructions sequentially following a near indirect JMP instruction (i.e., those not at the target) may be executed speculatively. If software needs to prevent this (e.g., in order to prevent a speculative execution side channel), then an INT3 or LFENCE instruction opcode can be placed after the near indirect JMP in order to block speculative execution.

Operation

```
IF near jump
  IF 64-bit Mode
    THEN
      IF near relative jump
        THEN
          tempRIP := RIP + DEST; (* RIP is instruction following JMP instruction*)
        ELSE (* Near absolute jump *)
          tempRIP := DEST;
      FI;
    ELSE
      IF near relative jump
        THEN
          tempEIP := EIP + DEST; (* EIP is instruction following JMP instruction*)
        ELSE (* Near absolute jump *)
          tempEIP := DEST;
      FI;
    FI;
  IF (IA32_EFER.LMA = 0 or target mode = Compatibility mode) and tempEIP outside code segment limit
    THEN #GP(0); FI
  IF 64-bit mode and tempRIP is not canonical
    THEN #GP(0);
  FI;
  IF OperandSize = 32
    THEN
      EIP := tempEIP;
    ELSE
      IF OperandSize = 16
        THEN (* OperandSize = 16 *)
          EIP := tempEIP AND 0000FFFFH;
        ELSE (* OperandSize = 64)
```

```

        RIP := tempRIP;
    FI;
FI;
IF (JMP near indirect, absolute indirect)
    IF EndbranchEnabledAndNotSuppressed(CPL)
        IF CPL = 3
            THEN
                IF ( no 3EH prefix OR IA32_U_CET.NO_TRACK_EN == 0 )
                    THEN
                        IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
                    FI;
                ELSE
                    IF ( no 3EH prefix OR IA32_S_CET.NO_TRACK_EN == 0 )
                        THEN
                            IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
                        FI;
                    FI;
                FI;
            FI;
        FI;
    FI;
FI;
IF far jump and (PE = 0 or (PE = 1 AND VM = 1)) (* Real-address or virtual-8086 mode *)
    THEN
        tempEIP := DEST(Offset); (* DEST is ptr16:32 or [m16:32] *)
        IF tempEIP is beyond code segment limit
            THEN #GP(0); FI;
        CS := DEST(segment selector); (* DEST is ptr16:32 or [m16:32] *)
        IF OperandSize = 32
            THEN
                EIP := tempEIP; (* DEST is ptr16:32 or [m16:32] *)
            ELSE (* OperandSize = 16 *)
                EIP := tempEIP AND 0000FFFFH; (* Clear upper 16 bits *)
            FI;
        FI;
    FI;
IF far jump and (PE = 1 and VM = 0)
    (* IA-32e mode or protected mode, not virtual-8086 mode *)
    THEN
        IF effective address in the CS, DS, ES, FS, GS, or SS segment is illegal or segment selector in target operand NULL
            THEN #GP(0); FI;
        IF segment selector index not within descriptor table limits
            THEN #GP(new selector); FI;
        Read type and access rights of segment descriptor;
        IF (IA32_EFER.LMA = 0)
            THEN
                IF segment type is not a conforming or nonconforming code segment, call gate, task gate, or TSS
                    THEN #GP(segment selector); FI;
            ELSE
                IF segment type is not a conforming or nonconforming code segment or call gate
                    THEN #GP(segment selector); FI;
            FI;
        Depending on type and access rights:
        GO TO CONFORMING-CODE-SEGMENT;
        GO TO NONCONFORMING-CODE-SEGMENT;
        GO TO CALL-GATE;
        GO TO TASK-GATE;
    FI;

```

```

        GO TO TASK-STATE-SEGMENT;
ELSE
    #GP(segment selector);
FI;
CONFORMING-CODE-SEGMENT:
    IF L-Bit = 1 and D-BIT = 1 and IA32_EFER.LMA = 1
        THEN GP(new code segment selector); FI;
    IF DPL > CPL
        THEN #GP(segment selector); FI;
    IF CR4.FRED = 1 and CPL = 0 and L-bit = 0
        THEN GP(new code segment selector); FI;
    IF segment not present
        THEN #NP(segment selector); FI;
    tempEIP := DEST(Offset);
    IF OperandSize = 16
        THEN tempEIP := tempEIP AND 0000FFFFH;
    FI;
    IF (IA32_EFER.LMA = 0 or target mode = Compatibility mode) and
    tempEIP outside code segment limit
        THEN #GP(0); FI
    IF tempEIP is non-canonical
        THEN #GP(0); FI;
    IF ShadowStackEnabled(CPL)
        IF (IA32_EFER.LMA and DEST(segment selector).L) = 0
            (* If target is legacy or compatibility mode then the SSP must be in low 4GB *)
            IF (SSP & 0xFFFFFFFF00000000 != 0)
                THEN #GP(0); FI;
        FI;
    FI;
    CS := DEST[segment selector]; (* Segment descriptor information also loaded *)
    CS(RPL) := CPL
    EIP := tempEIP;
    IF EndbranchEnabled(CPL)
        IF CPL = 3
            THEN
                IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
                IA32_U_CET.SUPPRESS = 0
            ELSE
                IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
                IA32_S_CET.SUPPRESS = 0
        FI;
    FI;
END;
NONCONFORMING-CODE-SEGMENT:
    IF L-Bit = 1 and D-BIT = 1 and IA32_EFER.LMA = 1
        THEN GP(new code segment selector); FI;
    IF (RPL > CPL) OR (DPL ≠ CPL)
        THEN #GP(code segment selector); FI;
    IF CR4.FRED = 1 and CPL = 0 and L-bit = 0
        THEN GP(new code segment selector); FI;
    IF segment not present
        THEN #NP(segment selector); FI;
    tempEIP := DEST(Offset);
    IF OperandSize = 16

```

```

    THEN tempEIP := tempEIP AND 0000FFFFH; FI;
IF (IA32_EFER.LMA = 0 OR target mode = Compatibility mode)
and tempEIP outside code segment limit
    THEN #GP(0); FI
IF tempEIP is non-canonical THEN #GP(0); FI;
IF ShadowStackEnabled(CPL)
    IF (IA32_EFER.LMA and DEST(segment selector).L) = 0
        (* If target is legacy or compatibility mode then the SSP must be in low 4GB *)
        IF (SSP & 0xFFFFFFFF00000000 != 0)
            THEN #GP(0); FI;
        FI;
    FI;
FI;
CS := DEST[segment selector]; (* Segment descriptor information also loaded *)
CS(RPL) := CPL;
EIP := tempEIP;
IF EndbranchEnabled(CPL)
    IF CPL = 3
        THEN
            IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_U_CET.SUPPRESS = 0
        ELSE
            IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
            IA32_S_CET.SUPPRESS = 0
        FI;
    FI;
FI;
END;

```

CALL-GATE:

```

IF call gate DPL < CPL or call gate DPL < call gate segment-selector RPL or CR4.FRED = 1
    THEN #GP(call gate selector); FI;
IF call gate not present
    THEN #NP(call gate selector); FI;
IF call gate code-segment selector is NULL
    THEN #GP(0); FI;
IF call gate code-segment selector index outside descriptor table limits
    THEN #GP(code segment selector); FI;
Read code segment descriptor;
IF code-segment segment descriptor does not indicate a code segment
or code-segment segment descriptor is conforming and DPL > CPL
or code-segment segment descriptor is non-conforming and DPL ≠ CPL
    THEN #GP(code segment selector); FI;
IF IA32_EFER.LMA = 1 and (code-segment descriptor is not a 64-bit code segment
or code-segment segment descriptor has both L-Bit and D-bit set)
    THEN #GP(code segment selector); FI;
IF code segment is not present
    THEN #NP(code-segment selector); FI;
tempEIP := DEST(Offset);
IF GateSize = 16
    THEN tempEIP := tempEIP AND 0000FFFFH; FI;
IF (IA32_EFER.LMA = 0 OR target mode = Compatibility mode) AND tempEIP
outside code segment limit
    THEN #GP(0); FI
CS := DEST[SegmentSelector]; (* Segment descriptor information also loaded *)
CS(RPL) := CPL;

```

```

EIP := tempEIP;
IF EndbranchEnabled(CPL)
    IF CPL = 3
        THEN
            IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH;
            IA32_U_CET.SUPPRESS = 0
        ELSE
            IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH;
            IA32_S_CET.SUPPRESS = 0
    FI;
FI;
END;
TASK-GATE:
    IF task gate DPL < CPL
    or task gate DPL < task gate segment-selector RPL
        THEN #GP(task gate selector); FI;
    IF task gate not present
        THEN #NP(gate selector); FI;
    Read the TSS segment selector in the task-gate descriptor;
    IF TSS segment selector local/global bit is set to local
    or index not within GDT limits
    or descriptor is not a TSS segment
    or TSS descriptor specifies that the TSS is busy
        THEN #GP(TSS selector); FI;
    IF TSS not present
        THEN #NP(TSS selector); FI;
    SWITCH-TASKS to TSS;
    IF EIP not within code segment limit
        THEN #GP(0); FI;
END;
TASK-STATE-SEGMENT:
    IF TSS DPL < CPL
    or TSS DPL < TSS segment-selector RPL
    or TSS descriptor indicates TSS not available
        THEN #GP(TSS selector); FI;
    IF TSS is not present
        THEN #NP(TSS selector); FI;
    SWITCH-TASKS to TSS;
    IF EIP not within code segment limit
        THEN #GP(0); FI;
END;

```

Flags Affected

All flags are affected if a task switch occurs; no flags are affected if a task switch does not occur.

Protected Mode Exceptions

#GP(0)	If offset in target operand, call gate, or TSS is beyond the code segment limits. If the segment selector in the destination operand, call gate, task gate, or TSS is NULL. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector. If target mode is compatibility mode and SSP is not in low 4GB.

#GP(selector)	If the segment selector index is outside descriptor table limits. If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, call gate, task gate, or task state segment. If the DPL for a nonconforming-code segment is not equal to the CPL (When not using a call gate.) If the RPL for the segment's segment selector is greater than the CPL. If the DPL for a conforming-code segment is greater than the CPL. If the DPL from a call-gate, task-gate, or TSS segment descriptor is less than the CPL or than the RPL of the call-gate, task-gate, or TSS's segment selector. If the segment descriptor for selector in a call gate does not indicate it is a code segment. If the segment descriptor for the segment selector in a task gate does not indicate an available TSS. If the segment selector for a TSS has its local/global bit set for local. If a TSS segment descriptor specifies that the TSS is busy or not available.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#NP (selector)	If the code segment being accessed is not present. If call gate, task gate, or TSS not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3. (Only occurs when fetching target from memory.)
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If the target operand is beyond the code segment limits. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made. (Only occurs when fetching target from memory.)
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same as 64-bit mode exceptions.

#GP(Selector)	If FRED transitions are enabled and the selector references a call gate.

64-Bit Mode Exceptions

#GP(0)	If a memory address is non-canonical. If target offset in destination operand is non-canonical. If target offset in destination operand is beyond the new code segment limit. If the segment selector in the destination operand is NULL. If the code segment selector in the 64-bit gate is NULL. If transitioning to compatibility mode and the SSP is beyond 4GB.

#GP(selector)	If the code segment or 64-bit call gate is outside descriptor table limits. If the code segment or 64-bit call gate overlaps non-canonical space. If the segment descriptor from a 64-bit call gate is in non-canonical space. If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, 64-bit call gate. If the segment descriptor pointed to by the segment selector in the destination operand is a code segment, and has both the D-bit and the L-bit set. If the DPL for a nonconforming-code segment is not equal to the CPL, or the RPL for the segment's segment selector is greater than the CPL. If the DPL for a conforming-code segment is greater than the CPL. If the DPL from a 64-bit call-gate is less than the CPL or than the RPL of the 64-bit call-gate. If the upper type field of a 64-bit call gate is not 0x0. If the segment selector from a 64-bit call gate is beyond the descriptor table limits. If the code segment descriptor pointed to by the selector in the 64-bit gate doesn't have the L-bit set and the D-bit clear. If the segment descriptor for a segment selector from the 64-bit call gate does not indicate it is a code segment. If the code segment is non-conforming and $CPL \neq DPL$. If the code segment is confirming and $CPL < DPL$. If FRED transitions are enabled and the selector references a call gate. If FRED transitions are enabled, CPL is 0, and the instruction would enter compatibility mode.
#NP(selector)	If a code segment or 64-bit call gate is not present.
#UD	(64-bit mode only) If a far jump is direct to an absolute address in memory. If the LOCK prefix is used.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

LDDQU—Load Unaligned Integer 128 Bits

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
F2 0F F0 /r LDDQU xmm1, mem	RM	V/V	SSE3	Load unaligned data from mem and return double quadword in xmm1.
VEX.128.F2.0F.WIG F0 /r VLDDQU xmm1, m128	RM	V/V	AVX	Load unaligned packed integer values from mem to xmm1.
VEX.256.F2.0F.WIG F0 /r VLDDQU ymm1, m256	RM	V/V	AVX	Load unaligned packed integer values from mem to ymm1.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

The instruction is *functionally similar* to (V)MOVDQU ymm/xmm, m256/m128 for loading from memory. That is: 32/16 bytes of data starting at an address specified by the source memory operand (second operand) are fetched from memory and placed in a destination register (first operand). The source operand need not be aligned on a 32/16-byte boundary. Up to 64/32 bytes may be loaded from memory; this is implementation dependent.

This instruction may improve performance relative to (V)MOVDQU if the source operand crosses a cache line boundary. In situations that require the data loaded by (V)LDDQU be modified and stored to the same location, use (V)MOVDQU or (V)MOVDQA instead of (V)LDDQU. To move a double quadword to or from memory locations that are known to be aligned on 16-byte boundaries, use the (V)MOVDQA instruction.

Implementation Notes

- If the source is aligned to a 32/16-byte boundary, based on the implementation, the 32/16 bytes may be loaded more than once. For that reason, the usage of (V)LDDQU should be avoided when using uncached or write-combining (WC) memory regions. For uncached or WC memory regions, keep using (V)MOVDQU.
- This instruction is a replacement for (V)MOVDQU (load) in situations where cache line splits significantly affect performance. It should not be used in situations where store-load forwarding is performance critical. If performance of store-load forwarding is critical to the application, use (V)MOVDQA store-load pairs when data is 256/128-bit aligned or (V)MOVDQU store-load pairs when data is 256/128-bit unaligned.
- If the memory address is not aligned on 32/16-byte boundary, some implementations may load up to 64/32 bytes and return 32/16 bytes in the destination. Some processor implementations may issue multiple loads to access the appropriate 32/16 bytes. Developers of multi-threaded or multi-processor software should be aware that on these processors the loads will be performed in a non-atomic way.
- If alignment checking is enabled (CR0.AM = 1, RFLAGS.AC = 1, and CPL = 3), an alignment-check exception (#AC) may or may not be generated (depending on processor implementation) when the memory address is not aligned on an 8-byte boundary.

In 64-bit mode, use of the REX.R prefix permits this instruction to access additional registers (XMM8-XMM15).

Note: In VEX-encoded versions, VEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

Operation

LDDQU (128-bit Legacy SSE Version)

DEST[127:0] := SRC[127:0]

DEST[MAXVL-1:128] (Unmodified)

VLDDQU (VEX.128 Encoded Version)

DEST[127:0] := SRC[127:0]

DEST[MAXVL-1:128] := 0

VLDDQU (VEX.256 Encoded Version)

DEST[255:0] := SRC[255:0]

DEST[MAXVL-1:256] := 0

Intel C/C++ Compiler Intrinsic Equivalent

LDDQU __m128i _mm_lddqu_si128 (__m128i * p);

VLDDQU __m256i _mm256_lddqu_si256 (__m256i * p);

Numeric Exceptions

None.

Other Exceptions

See Table 2-21, “Type 4 Class Exception Conditions.”

Note treatment of #AC varies.

LKGS—Load Kernel GS Base

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
F2 0F 00 /6	LKGS r/m16	A	Valid	N.E.	Load GS using r/m16, placing the base address instead in the IA32_KERNEL_GS_BASE MSR.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	16-bit selector	N/A	N/A	N/A

Description

LKGS operates in the same way as MOV to GS except that the descriptor's base address is loaded into the IA32_KERNEL_GS_BASE MSR instead of the GS segment's descriptor cache.

LKGS takes a single (source) operand, which can be a general-purpose register or a memory location. The operand must be a valid segment selector. The instruction loads the segment descriptor referenced by that segment selector into the GS descriptor cache, with the exception of the base address. The base address in the GS descriptor cache is not modified; the base address from the segment descriptor is loaded into the IA32_KERNEL_GS_BASE MSR. (Since the base address in the descriptor is only 32 bits, the upper 32 bits of the MSR are cleared.)

A null segment selector (values 0000-0003) can be loaded without causing an exception. However, any subsequent attempt to reference GS outside 64-bit mode causes a general protection exception (#GP) and no memory reference occurs. LKGS with a null segment selector loads zero into IA32_KERNEL_GS_BASE.

Operation

IF CPL > 0 OR logical processor not in 64-bit mode

THEN #UD; FI;

IF SRC is null

THEN

GS.selector := SRC;

mark GS as null;

IA32_KERNEL_GS_BASE := 0;

ELSE

IF SRC.index is outside descriptor table limits

THEN #GP(selector); FI;

read referenced descriptor for descriptor table;

IF the descriptor is not for a data or readable code segment OR SRC.RPL > descriptor.DPL

THEN #GP(selector); FI;

IF the descriptor is not marked present

THEN #NP(selector);

ELSE

GS.selector := SRC;

GS.attributes := descriptor.attributes;

IA32_KERNEL_GS_BASE := descriptor.base; // bits 63:32 cleared

FI;

FI;

Flags Affected

None.

Protected Mode Exceptions

#UD The LKGS instruction is not recognized in protected mode.

Real-Address Mode Exceptions

#UD The LKGS instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The LKGS instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD The LKGS instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD If CPL > 0.
 If CPUID.07H.01H:EAX.LKGS[18] is not 1.
 If the LOCK prefix is used.

#GP(0) If the memory address is in a non-canonical form.

#GP(selector) If segment selector index is outside descriptor table limits.
 If the memory access to the descriptor table is non-canonical.
 If the referenced descriptor is not for a data or readable code segment or the selector RPL is greater than the descriptor DPL.

#SS(0) If the memory address is in a non-canonical form.

#PF(fault-code) If a page fault occurs.

7. Updates to Chapter 4, Volume 2B

Change bars and text show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, M-U*.

Changes to this chapter:

- Added new instruction PBNDKB and updated instruction PCONFIG for Total Storage Encryption.
- Updated instruction PSHUFB clarifying descriptions for its VEX and EVEX forms in the Opcode table. The Description and Operation (pseudocode) sections were also updated.
- Duplicate paragraph in the PSRLDQ instruction was removed.
- Redundant conditional pseudocode in the REP instruction was removed.
- The encoding in the opcode table for instruction SAVEPREVSSP was simplified.
- The following instructions were updated for Flexible Return Event Delivery (FRED):
 - RET
 - SETSSBSY
 - SWAPGS
 - SYSCALL
 - SYSENTER
 - SYSEXIT
 - SYSRET

4.1 IMM8 CONTROL BYTE OPERATION FOR PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMPISTRM

The notations introduced in this section are referenced in the reference pages of PCMPESTRI, PCMPESTRM, PCMPISTRI, PCMPISTRM. The operation of the immediate control byte is common to these four string text processing instructions of SSE4.2. This section describes the common operations.

4.1.1 General Description

The operation of PCMPESTRI, PCMPESTRM, PCMPISTRI, PCMPISTRM is defined by the combination of the respective opcode and the interpretation of an immediate control byte that is part of the instruction encoding.

The opcode controls the relationship of input bytes/words to each other (determines whether the inputs terminated strings or whether lengths are expressed explicitly) as well as the desired output (index or mask).

The imm8 control byte for PCMPESTRM/PCMPESTRI/PCMPISTRM/PCMPISTRI encodes a significant amount of programmable control over the functionality of those instructions. Some functionality is unique to each instruction while some is common across some or all of the four instructions. This section describes functionality which is common across the four instructions.

The arithmetic flags (ZF, CF, SF, OF, AF, PF) are set as a result of these instructions. However, the meanings of the flags have been overloaded from their typical meanings in order to provide additional information regarding the relationships of the two inputs.

PCMPxSTRx instructions perform arithmetic comparisons between all possible pairs of bytes or words, one from each packed input source operand. The boolean results of those comparisons are then aggregated in order to produce meaningful results. The imm8 control byte is used to affect the interpretation of individual input elements as well as control the arithmetic comparisons used and the specific aggregation scheme.

Specifically, the imm8 Control Byte consists of bit fields that control the following attributes:

- **Source data format** — Byte/word data element granularity, signed or unsigned elements.
- **Aggregation operation** — Encodes the mode of per-element comparison operation and the aggregation of per-element comparisons into an intermediate result.
- **Polarity** — Specifies intermediate processing to be performed on the intermediate result.
- **Output selection** — Specifies final operation to produce the output (depending on index or mask) from the intermediate result.

4.1.2 Source Data Format

Table 4-1. Source Data Format

Imm8[1:0]	Meaning	Description
00b	Unsigned bytes	Both 128-bit sources are treated as packed, unsigned bytes.
01b	Unsigned words	Both 128-bit sources are treated as packed, unsigned words.
10b	Signed bytes	Both 128-bit sources are treated as packed, signed bytes.
11b	Signed words	Both 128-bit sources are treated as packed, signed words.

If the imm8 control byte has bit[0] cleared, each source contains 16 packed bytes. If the bit is set each source contains 8 packed words. If the imm8 control byte has bit[1] cleared, each input contains unsigned data. If the bit is set each source contains signed data.

4.1.3 Aggregation Operation

Table 4-2. Aggregation Operation

Imm8[3:2]	Mode	Comparison
00b	Equal any	The arithmetic comparison is “equal.”
01b	Ranges	Arithmetic comparison is “greater than or equal” between even indexed bytes/words of reg and each byte/word of reg/mem. Arithmetic comparison is “less than or equal” between odd indexed bytes/words of reg and each byte/word of reg/mem. (reg/mem[m] >= reg[n] for n = even, reg/mem[m] <= reg[n] for n = odd)
10b	Equal each	The arithmetic comparison is “equal.”
11b	Equal ordered	The arithmetic comparison is “equal.”

All 256 (64) possible comparisons are always performed. The individual Boolean results of those comparisons are referred by “BoolRes[Reg/Mem element index, Reg element index].” Comparisons evaluating to “True” are represented with a 1, False with a 0 (positive logic). The initial results are then aggregated into a 16-bit (8-bit) intermediate result (IntRes1) using one of the modes described in the table below, as determined by imm8 control byte bits[3:2].

See Section 4.1.6 for a description of the overrideIfDataInvalid() function used in Table 4-3.

Table 4-3. Aggregation Operation

Mode	Pseudocode
Equal any (find characters from a set)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For j = 0 to UpperBound, j++ For i = 0 to UpperBound, i++ IntRes1[j] OR= overrideIfDataInvalid(BoolRes[j,i])
Ranges (find characters from ranges)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For j = 0 to UpperBound, j++ For i = 0 to UpperBound, i+=2 IntRes1[j] OR= (overrideIfDataInvalid(BoolRes[j,i]) AND overrideIfDataInvalid(BoolRes[j,i+1]))
Equal each (string compare)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For i = 0 to UpperBound, i++ IntRes1[i] = overrideIfDataInvalid(BoolRes[i,i])
Equal ordered (substring search)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = imm8[0] ? FFH : FFFFH For j = 0 to UpperBound, j++ For i = 0 to UpperBound-j, k=j to UpperBound, k++, i++ IntRes1[j] AND= overrideIfDataInvalid(BoolRes[k,i])

4.1.4 Polarity

IntRes1 may then be further modified by performing a 1’s complement, according to the value of the imm8 control byte bit[4]. Optionally, a mask may be used such that only those IntRes1 bits which correspond to “valid” reg/mem input elements are complemented (note that the definition of a valid input element is dependent on the specific opcode and is defined in each opcode’s description). The result of the possible negation is referred to as IntRes2.

Table 4-4. Polarity

Imm8[5:4]	Operation	Description
00b	Positive Polarity (+)	IntRes2 = IntRes1
01b	Negative Polarity (-)	IntRes2 = -1 XOR IntRes1
10b	Masked (+)	IntRes2 = IntRes1
11b	Masked (-)	IntRes2[i] = IntRes1[i] if reg/mem[i] invalid, else = ~IntRes1[i]

4.1.5 Output Selection

Table 4-5. Output Selection

Imm8[6]	Operation	Description
0b	Least significant index	The index returned to ECX is of the least significant set bit in IntRes2.
1b	Most significant index	The index returned to ECX is of the most significant set bit in IntRes2.

For PCMPESTRI/PCMPISTRI, the imm8 control byte bit[6] is used to determine if the index is of the least significant or most significant bit of IntRes2.

Table 4-6. Output Selection

Imm8[6]	Operation	Description
0b	Bit mask	IntRes2 is returned as the mask to the least significant bits of XMM0 with zero extension to 128 bits.
1b	Byte/word mask	IntRes2 is expanded into a byte/word mask (based on imm8[1]) and placed in XMM0. The expansion is performed by replicating each bit into all of the bits of the byte/word of the same index.

Specifically for PCMPESTRM/PCMPISTRM, the imm8 control byte bit[6] is used to determine if the mask is a 16 (8) bit mask or a 128 bit byte/word mask.

4.1.6 Valid/Invalid Override of Comparisons

PCMPxSTRx instructions allow for the possibility that an end-of-string (EOS) situation may occur within the 128-bit packed data value (see the instruction descriptions below for details). Any data elements on either source that are determined to be past the EOS are considered to be invalid, and the treatment of invalid data within a comparison pair varies depending on the aggregation function being performed.

In general, the individual comparison result for each element pair BoolRes[i..j] can be forced true or false if one or more elements in the pair are invalid. See Table 4-7.

Table 4-7. Comparison Result for Each Element Pair BoolRes[i..j]

xmm1 byte/ word	xmm2/ m128 byte/word	Imm8[3:2] = 00b (equal any)	Imm8[3:2] = 01b (ranges)	Imm8[3:2] = 10b (equal each)	Imm8[3:2] = 11b (equal ordered)
Invalid	Invalid	Force false	Force false	Force true	Force true
Invalid	Valid	Force false	Force false	Force false	Force true
Valid	Invalid	Force false	Force false	Force false	Force false
Valid	Valid	Do not force	Do not force	Do not force	Do not force

4.1.7 Summary of Im8 Control byte

Table 4-8. Summary of Imm8 Control Byte

Imm8	Description
-----0b	128-bit sources treated as 16 packed bytes.
-----1b	128-bit sources treated as 8 packed words.
-----0-b	Packed bytes/words are unsigned.
-----1-b	Packed bytes/words are signed.
---00--b	Mode is equal any.
---01--b	Mode is ranges.
---10--b	Mode is equal each.
---11--b	Mode is equal ordered.
---0---b	IntRes1 is unmodified.
---1---b	IntRes1 is negated (1's complement).
--0----b	Negation of IntRes1 is for all 16 (8) bits.
--1----b	Negation of IntRes1 is masked by reg/mem validity.
-0-----b	Index of the least significant, set, bit is used (regardless of corresponding input element validity). IntRes2 is returned in least significant bits of XMM0.
-1-----b	Index of the most significant, set, bit is used (regardless of corresponding input element validity). Each bit of IntRes2 is expanded to byte/word.
0-----b	This bit currently has no defined effect, should be 0.
1-----b	This bit currently has no defined effect, should be 0.

4.1.8 Diagram Comparison and Aggregation Process

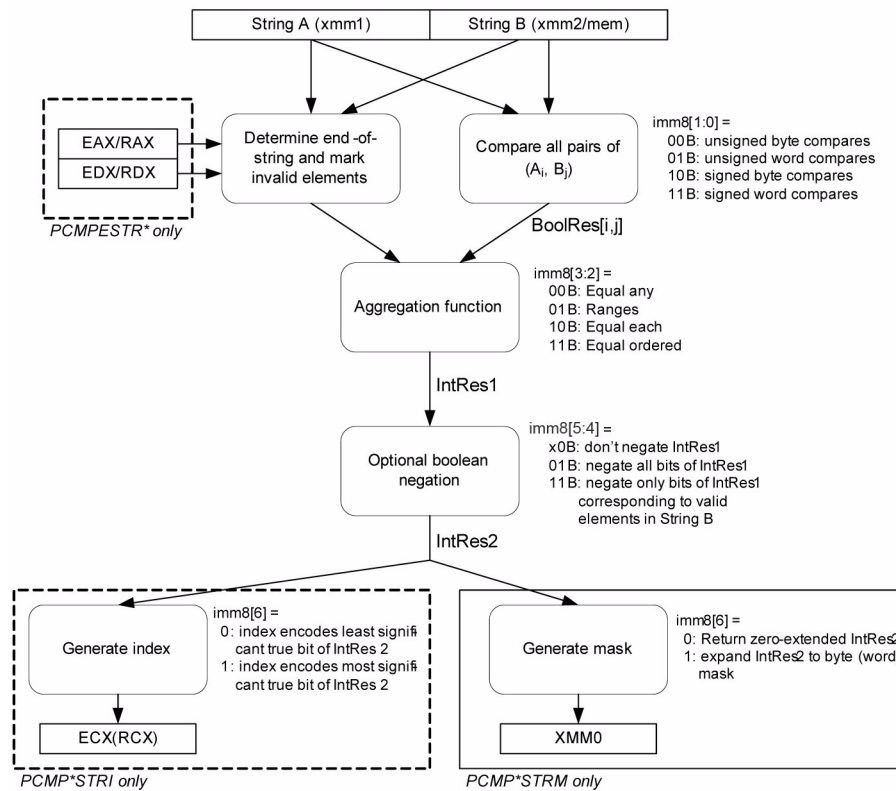


Figure 4-1. Operation of PCMPSTRx and PCMPSTRx

4.2 COMMON TRANSFORMATION AND PRIMITIVE FUNCTIONS FOR SHA1XXX AND SHA256XXX

The following primitive functions and transformations are used in the algorithmic descriptions of SHA1 and SHA256 instruction extensions SHA1NEXTE, SHA1RND4, SHA1MSG1, SHA1MSG2, SHA256RND4, SHA256MSG1, and SHA256MSG2. The operands of these primitives and transformation are generally 32-bit DWORD integers.

- **f0():** A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 1 to 20 processing.

$$f0(B,C,D) := (B \text{ AND } C) \text{ XOR } ((\text{NOT}(B) \text{ AND } D))$$
- **f1():** A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 21 to 40 processing.

$$f1(B,C,D) := B \text{ XOR } C \text{ XOR } D$$
- **f2():** A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 41 to 60 processing.

$$f2(B,C,D) := (B \text{ AND } C) \text{ XOR } (B \text{ AND } D) \text{ XOR } (C \text{ AND } D)$$
- **f3():** A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 61 to 80 processing. It is the same as f1().

$$f3(B,C,D) := B \text{ XOR } C \text{ XOR } D$$
- **Ch():** A bit oriented logical operation that derives a new dword from three SHA256 state variables (dword).

$\text{Ch}(E,F,G) := (E \text{ AND } F) \text{ XOR } ((\text{NOT } E) \text{ AND } G)$

- $\text{Maj}()$: A bit oriented logical operation that derives a new dword from three SHA256 state variables (dword).
 $\text{Maj}(A,B,C) := (A \text{ AND } B) \text{ XOR } (A \text{ AND } C) \text{ XOR } (B \text{ AND } C)$

ROR is rotate right operation

$(A \text{ ROR } N) := A[N-1:0] \parallel A[\text{Width}-1:N]$

ROL is rotate left operation

$(A \text{ ROL } N) := A \text{ ROR } (\text{Width}-N)$

SHR is the right shift operation

$(A \text{ SHR } N) := \text{ZEROES}[N-1:0] \parallel A[\text{Width}-1:N]$

- $\Sigma_0()$: A bit oriented logical and rotational transformation performed on a dword SHA256 state variable.
 $\Sigma_0(A) := (A \text{ ROR } 2) \text{ XOR } (A \text{ ROR } 13) \text{ XOR } (A \text{ ROR } 22)$
- $\Sigma_1()$: A bit oriented logical and rotational transformation performed on a dword SHA256 state variable.
 $\Sigma_1(E) := (E \text{ ROR } 6) \text{ XOR } (E \text{ ROR } 11) \text{ XOR } (E \text{ ROR } 25)$
- $\sigma_0()$: A bit oriented logical and rotational transformation performed on a SHA256 message dword used in the message scheduling.
 $\sigma_0(W) := (W \text{ ROR } 7) \text{ XOR } (W \text{ ROR } 18) \text{ XOR } (W \text{ SHR } 3)$
- $\sigma_1()$: A bit oriented logical and rotational transformation performed on a SHA256 message dword used in the message scheduling.
 $\sigma_1(W) := (W \text{ ROR } 17) \text{ XOR } (W \text{ ROR } 19) \text{ XOR } (W \text{ SHR } 10)$
- K_i : SHA1 Constants dependent on immediate i.
 $K0 = 0x5A827999$
 $K1 = 0x6ED9EBA1$
 $K2 = 0X8F1BBCDC$
 $K3 = 0xCA62C1D6$

4.3 INSTRUCTIONS (M-U)

Chapter 4 continues an alphabetical discussion of Intel® 64 and IA-32 instructions (M-U). See also: Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A; Chapter 4, "Instruction Set Reference, M-U," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C; and Chapter 4, "Instruction Set Reference, M-U," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D.

PBNDKB—Platform Bind Key to Binary Large Object

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 C7 PBNDKB	Z0	V/I	PBNDKB	This instruction is used to bind information to a platform by encrypting it with a platform-specific wrapping key.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

The PBNDKB instruction allows software to bind information to a platform by encrypting it with a platform-specific wrapping key. The encrypted data may later be used by the PCONFIG instruction to configure the total storage encryption (TSE) engine.

The instruction can be executed only in 64-bit mode. The registers RBX and RCX provide input information to the instruction. Executions of PBNDKB may fail for platform-specific reasons. An execution reports failure by setting the ZF flag and loading EAX with a non-zero failure reason; a successful execution clears ZF and EAX.

The instruction operates on 256-byte data structures called **bind structures**. It reads a bind structure at the linear address in RBX and writes a modified bind structure to the linear address in RCX. The addresses in RBX and RCX must be different from each other and must be 256-byte aligned.

The instruction encrypts a portion of the input bind structure and generates a MAC of parts of that structure. The encrypted data and MAC are written out as part of the output bind structure.

The format of a bind structure is given in Table 4-13.

Table 4-13. Bind Structure Format

Field	Offset (bytes)	Size (bytes)	Comments
MAC	0	16	Output by PBNDKB as a MAC based on the input bind structure
Reserved	16	8	Reserved; must be zero on input, output as zero
IV	24	12	Initialization vector generated and output by PBNDKB
Reserved	36	28	Reserved; must be zero on input, output as zero
BTENCDATA	64	64	Encryption data (plaintext on input; ciphertext on output)
BTDATA	128	128	Additional control and data (modified but not encrypted)

A description of each of the fields in a bind structure is provided below:

- **MAC:** A MAC produced by PBNDKB of parts of its input bind structure. This field in the input bind structure is not used.
- **IV:** PBNDKB randomly generates a 96-bit initialization vector and uses it as input to an authenticated encryption function. The generated IV is written to the output bind structure. If there is insufficient entropy for the random-number generator, PBNDKB will fail and report the failure by loading EAX with value 1 (ENTROPY_ERROR). This field in the input bind structure is not used.
- **BTENCDATA:** In the input bind structure, the field contains the data to be encrypted. The data consist of two 256-bit keys, a data key and a tweak key. If the value of the KEY_GENERATION_CTRL field of the BTDATA (see below) is 1, PBNDKB randomizes the values of these keys before encrypting them. (If there is insufficient entropy for the random-number generator, PBNDKB will fail and report the failure by loading EAX with value 1 (ENTROPY_ERROR).) PBNDKB writes the encrypted data to this field in the output bind structure.
- **BTDATA:** This field contains additional control and data that are not encrypted. It has the following format:

- **USER_SUPP_CHALLENGE** (bytes 31:0): PBNDKB uses this value in the input bind structure to determine the wrapping key (see below). It writes zero to this field in the output bind structure.
- **KEY_GENERATION_CTRL** (byte 32): PBNDKB uses this value in the input bind structure to determine whether to randomize the keys being encrypted. The value must be 0 or 1 (otherwise, a #GP occurs).
- The remaining 95 bytes are reserved and must be zero.

PBNDKB determines a 256-bit **wrapping key** by computing an HMAC based on SHA-256 using 256-bit platform-specific key and the USER_SUPP_CHALLENGE in the BTDATA field in the input bind structure.

PBNDKB then uses the wrapping key and an AES GCM authenticated encryption function to encrypt BTENCDATA and produce a MAC. The encryption function uses the following inputs:

- The 64-byte BTENCDATA to be encrypted (which may have been randomized; see above).
- The 256-bit wrapping key.
- The 96-bit IV randomly generated by PBNDKB.
- 176 bytes of additional authenticated data that are the concatenation of 8 bytes of zeroes, the IV, 28 bytes of zeroes, and the BTDATA in the input bind structure.
- The length of the additional authenticated data (176).

The encryption function produces a structure with 64 bytes of encrypted data and a 16-byte MAC. PBNDKB saves these values to the corresponding fields in its output bind structure. Other fields are copied from the input bind structure or written as zero, except the IV (which receives the randomly generated value) and the USER_SUPP_CHALLENGE in the BTDATA, which is written as zero.

Operation

(* #UD if PBNDKB is not enumerated, CPL > 0, or not in 64-bit mode*)

```
IF CPUID.(EAX=07H, ECX=01H):EBX.PBNDKB[bit 1] = 0 OR CPL > 0 OR not in 64-bit mode
    THEN #UD; FI;
```

(* #GP if pointers are not aligned or overlapping *)

```
IF RBX = RCX OR RBX is not 256-byte aligned OR RCX is not 256-byte aligned
    THEN #GP(0); FI;
```

Load TMP_BIND_STRUCT from 256 bytes at linear address in RBX;

(* MAC and IV fields might not be read. *)

(* Check TMP_BIND_STRUCT for illegal values *)

```
IF bytes 23:16 and bytes 63:36 of TMP_BIND_STRUCT are not all zero
    THEN #GP(0); FI;
```

```
IF TMP_BIND_STRUCT.BTDATA.KEY_GENERATION_CTRL > 1
    THEN #GP(0); FI;
```

```
IF bytes 127:33 of TMP_BIND_STRUCT.BTDATA are not all zero
    THEN #GP(0); FI;
```

(* Randomize input keys if requested *)

```
IF TMP_BIND_STRUCT.BTDATA.KEY_GENERATION_CONTROL = 1
    THEN
```

```
    Load RNG_DATA_KEY with a random 256-bit value using hardware RNG;
```

```
    Load RNG_TWEAK_KEY with a random 256-bit value using hardware RNG;
```

```
    IF there was insufficient entropy
```

```
        THEN (* PBNDKB failure *)
```

```
            RFLAGS.ZF := 1;
```

```
            RAX := ENTROPY_ERROR; (* failure reason 1 *)
```

```
            GOTO EXIT;
```

```
    FI;
```

```

    (* XOR the input keys with the random keys; this does not modify input bind structure in memory *)
    TMP_BIND_STRUCT.BTENCDATA.DATA_KEY := RNG_DATA_KEY XOR TMP_BIND_STRUCT.BTENCDATA.DATA_KEY;
    TMP_BIND_STRUCT.BTENCDATA.TWEAK_KEY := RNG_TWEAK_KEY XOR TMP_BIND_STRUCT.BTENCDATA.TWEAK_KEY;
FI;

(* Compute wrapping key from platform key and user challenge *)
PLATFORM_KEY := 256-bit platform-specific key;
WRAPPING_KEY := HMAC_SHA256(PLATFORM_KEY, TMP_BIND_STRUCT.BTDATA.USER_SUPP_CHALLENGE);

(* Generate random data for initialization vector *)
Load TMP_IV with a random 96-bit value using hardware RNG;
IF there was insufficient entropy
    THEN (* PBNDKB failure *)
        RFLAGS.ZF := 1;
        RAX := ENTROPY_ERROR;  (* failure reason 1 *)
        GOTO EXIT;
FI;

(* Compose 176 bytes of additional authenticated data for use by authenticated decryption *)
AAD := Concatenation of 8 bytes of zeroes, TMP_IV, 28 bytes of zeroes, and TMP_BIND_STRUCT.BTDATA;

ENCRYPT_STRUCT := AES256_GCM_ENC(TMP_BIND_STRUCT.BTENCDATA, WRAPPING_KEY, TMP_IV, AAD, 176);

OUT_BIND_STRUCT.MAC := ENCRYPT_STRUCT.MAC;
OUT_BIND_STRUCT[bytes 23:16] := 0;
OUT_BIND_STRUCT.IV := TMP_IV;
OUT_BIND_STRUCT[bytes 63:36] := 0;
OUT_BIND_STRUCT.BTENCDATA := ENCRYPT_STRUCT.ENC_DATA;
OUT_BIND_STRUCT.BTDATA.USER_SUPP_CHALLENGE := 0;
OUT_BIND_STRUCT.BTDATA.KEY_GENERATION_CTRL := IN_BIND_STRUCT.BTDATA.KEY_GENERATION_CTRL;
OUT_BIND_STRUCT.BTDATA[bytes 127:33] := 0;

(* Save OUT_BIND_STRUCT to memory *)
Store OUT_BIND_STRUCT to 256 bytes at linear address in RCX;

(* Indicate successful completion *)
RAX := 0;
RFLAGS.ZF := 0;

EXIT:
RFLAGS.CF := 0;
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;

```

Protected Mode Exceptions

#UD PBNDKB is not supported in protected mode.

Real-Address Mode Exceptions

#UD PBNDKB is not supported in real-address mode.

Virtual-8086 Mode Exceptions

#UD PBNDKB is not supported in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

- #GP(0) If the values of RBX and RCX are not both canonical.
 If RBX or RCX is not 256B aligned.
 If RBX = RCX.
 If any of the reserved bytes in the input bind structure are set (including bytes in BTDATA).
 If the value of the key-generation control in the BTDATA field of the input bind structure is not 0 or 1.
- #PF(fault-code) If a page fault occurs in accessing memory operands.
- #UD If any of the LOCK/REP/Operand Size/VEX prefixes are used.
 If the current privilege level is not 0.
 If CPUID.07H.01H:EBX.PBNDKB[1] = 0.

PCONFIG—Platform Configuration

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 C5 PCONFIG	A	V/V	PCONFIG	This instruction is used to execute functions for configuring platform features.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	N/A	N/A	N/A	N/A

Description

The PCONFIG instruction allows software to configure certain platform features. It supports these features with multiple leaf functions, selecting a leaf function using the value in EAX.

Depending on the leaf function, the registers RBX, RCX, and RDX may be used to provide input information or for the instruction to report output information. Addresses and operands are 32 bits outside 64-bit mode and are 64 bits in 64-bit mode. The value of CS.D does not affect operand size or address size.

Executions of PCONFIG may fail for platform-specific reasons. An execution reports failure by setting the ZF flag and loading EAX with a non-zero failure reason; a successful execution clears ZF and EAX.

Each PCONFIG leaf function applies to a specific hardware block called a PCONFIG target. The leaf function is supported only if the processor supports that target. Each target is associated with a numerical target identifier, and CPUID leaf 1BH (PCONFIG information) enumerates the identifiers of the supported targets. An attempt to execute an undefined leaf function, or a leaf function that applies to an unsupported target identifier, results in a general-protection exception (#GP).

Leaf Function MKTME_KEY_PROGRAM

- PCONFIG leaf function 0 (selected by loading EAX with value 0) is used for key programming for total memory encryption-multi-key (TME-MK).¹ This leaf function is called MKTME_KEY_PROGRAM and it pertains to the TME-MK target, which has target identifier 1. The leaf function uses the EBX (or RBX) register for additional input information.
- Software uses this leaf function to manage the encryption key associated with a particular key identifier (KeyID). The leaf function uses a data structure called the **TME-MK key programming structure** (MKTME_KEY_PROGRAM_STRUCT). Software provides the address of the structure (as an offset in the DS segment) in EBX (or RBX). The format of the structure is given in Table 4-16.

Table 4-16. MKTME_KEY_PROGRAM_STRUCT Format

Field	Offset (bytes)	Size (bytes)	Comments
KEYID	0	2	Key Identifier.
KEYID_CTRL	2	4	KeyID control: ▪ Bits 7:0: key-programming command (COMMAND) ▪ Bits 23:8: encryption algorithm (ENC_ALG) ▪ Bits 31:24: Reserved, must be zero (RSVD)
Ignored	6	58	Not used.
KEY_FIELD_1	64	64	Software supplied data key or entropy for data key.
KEY_FIELD_2	128	64	Software supplied tweak key or entropy for tweak key.

1. Further details on TME-MK can be found here:

<https://software.intel.com/sites/default/files/managed/a5/16/Multi-Key-Total-Memory-Encryption-Spec.pdf>

A description of each of the fields in MKTME_KEY_PROGRAM_STRUCT is provided below:

- **KEYID:** The key identifier (KeyID) being programmed to the TME-MK engine. This leaf function causes a general-protection exception (#GP) if the KeyID is zero. KeyID zero always uses the current behavior configured for TME (total memory encryption), either to encrypt with platform TME key or to bypass TME encryption. This leaf function also causes a #GP if the KeyID exceeds the maximum enumerated IA32_TME_CAPABILITY.MK_TME_MAX_KEYS[50:36] or configured by the setting of IA32_TME_ACTIVATE.MK_TME_KEYID_BITS[39:32].
 - If IA32_TME_ACTIVATE.MK_TME_KEYID_BITS[35:32] = k , KEYID[15: k] must be zero.
 - If the logical processor is outside secure arbitration mode (SEAM; see Chapter 35) and IA32_TME_ACTIVATE.TDX_RESERVED_KEYID_BITS[39:36] = $p > 0$, KEYID[$k-1:k-p$] must also be zero. (Note that p will never exceed k .)

If either is not true, the leaf function causes #GP.

- **KEYID_CTRL:** The KEYID_CTRL field comprises two sub-fields used by software to control the encryption performed for the selected KeyID:
 - Key-programming command (COMMAND; bits 7:0). This 8-bit field should contain one of the following values:
 - KEYID_SET_KEY_DIRECT (value 0). With this command, software programs directly the encryption key to be used for the selected KeyID.
 - KEYID_SET_KEY_RANDOM (value 1). With this command, software has the CPU generate and assign an encryption key to be used for the selected KeyID using a hardware random-number generator.

If this command is used and there is insufficient entropy for the random-number generator, the leaf function will fail and report the failure by loading EAX with value 2 (ENTROPY_ERROR).

Because the keys programmed by this leaf function are discarded on reset and software cannot read the programmed keys, the keys programmed with this command are ephemeral.
 - KEYID_CLEAR_KEY (value 2). With this command, software indicates that the selected KeyID should use the current behavior configured for TME (see above).
 - KEYID_NO_ENCRYPT (value 3). With this command, software indicates that no encryption should be used for the selected KeyID.

If any other value is used, the leaf function causes a #GP.

- Encryption algorithm (ENC_ALG, bits 23:8). Bits 63:48 of the IA32_TME_ACTIVATE MSR (MSR index 982H) indicate which encryption algorithms are supported by the platform. The 16-bit ENC_ALG field should specify one of the algorithms indicated in IA32_TME_ACTIVATE. This leaf function causes a #GP if ENC_ALG does not set exactly one bit or if it sets a bit whose corresponding bit is not set in IA32_TME_ACTIVATE[63:48].
- **KEY_FIELD_1:** Use of this field depends upon selected key-programming command:
 - If the direct key-programming command is used (KEYID_SET_KEY_DIRECT), this field carries the software supplied data key to be used for the KeyID.
 - If the random key-programming command is used (KEYID_SET_KEY_RANDOM), this field carries the software supplied entropy to be mixed in the CPU generated random data key.
 - This field is ignored when one of the other key-programming commands is used.

It is software's responsibility to ensure that the key supplied for the direct key-programming option or the entropy supplied for the random key-programming option does not result in weak keys. There are no explicit checks in the instruction to detect or prevent weak keys.
- **KEY_FIELD_2:** Use of this field depends upon selected key-programming command:
 - If the direct key-programming command is used (KEYID_SET_KEY_DIRECT), this field carries the software supplied tweak key to be used for the KeyID.
 - If the random key-programming command is used (KEYID_SET_KEY_RANDOM), this field carries the software supplied entropy to be mixed in the CPU generated random tweak key.
 - This field is ignored when one of the other key-programming commands is used.

It is software's responsibility to ensure that the key supplied for the direct key-programming option or the entropy supplied for the random key-programming option does not result in weak keys. There are no explicit checks in the instruction to detect or prevent weak keys.

All KeyIDs default to TME behavior (encrypt with TME key or bypass encryption) on activation of TME-MK. Software can at any point decide to change the key for a KeyID using [this](#) leaf function. Changing the key for a KeyID does **not** change the state of the TLB caches or memory pipeline. Software is responsible for taking appropriate actions to ensure correct behavior.

The key table used by TME-MK is shared by all logical processors in a platform. For this reason, execution of the MKTME_KEY_PROGRAM leaf function must gain exclusive access to the key table before updating it. The leaf function does this by acquiring lock (implemented in the platform) and retaining that lock until the execution completes. An execution of the leaf function may fail to acquire the lock if it is already in use. In this situation, the leaf function will load EAX with failure reason 5 (DEVICE_BUSY) indicating that software must retry. When this happens, the key table is not updated, and software should retry execution of PCONFIG.

Leaf Function TSE_KEY_PROGRAM

PCONFIG leaf function 1(selected by loading EAX with value 1) is used for direct key programming for total storage encryption (TSE). This leaf function is called TSE_KEY_PROGRAM and it pertains to the TSE target, which has target identifier 2. The leaf function can be used only in 64-bit mode. It uses the RBX register for additional input information.

Software uses this leaf function to manage the encryption key associated with a particular key identifier (KeyID). The leaf function uses a data structure called the **TSE key programming structure** (TSE_KEY_PROGRAM_STRUCT). Software provides the linear address of the structure in RBX. The format of the structure is given in Table 4-17.

Table 4-17. TSE_KEY_PROGRAM_STRUCT Format

Field	Offset (bytes)	Size (bytes)	Comments
KEYID	0	2	Key Identifier.
KEYID_CTRL	2	4	KeyID control: - Bits 7:0: key-programming command (COMMAND) - Bits 23:8: encryption algorithm (ENC_ALG) - Bits 31:24: Reserved, must be zero (RSVD)
Ignored	6	58	Not used.
KEY_FIELD_1	64	64	Software supplied data key.
KEY_FIELD_2	128	64	Software supplied tweak key.

A description of each of the fields in MKTME_KEY_PROGRAM_STRUCT is provided below:

- **KEYID:** The key identifier (KeyID) being programmed to the TSE engine. The leaf function causes a general-protection exception (#GP) if the KeyID exceeds the maximum enumerated in the TSE_MAX_KEYS field (bits 50:36) of the IA32_TSE_CAPABILITY MSR (MSR index 9F1H).
- **KEYID_CTRL:** The KEYID_CTRL field comprises two sub-fields used by software to control the encryption performed for the selected KeyID:
 - Key-programming command (COMMAND; bits 7:0). This 8-bit field should contain one of the following values:
 - TSE_SET_KEY_DIRECT (value 0). With this command, software programs directly the encryption key to be used for the selected KeyID.
 - TSE_NO_ENCRYPT (value 1). With this command, software indicates that no encryption should be used for the selected KeyID.

If any other value is used, the leaf function causes a #GP.

- Encryption algorithm (ENC_ALG, bits 23:8). IA32_TSE_CAPABILITY[15:0] indicates which encryption algorithms are supported by the platform. The 16-bit ENC_ALG field should specify one of the algorithms

indicated in IA32_TSE_CAPABILITY. The leaf function causes a #GP if ENC_ALG does not set exactly one bit or if it sets a bit whose corresponding bit is not set in IA32_TSE_CAPABILITY.

- **KEY_FIELD_1:** If the direct key-programming command is used (TSE_SET_KEY_DIRECT), this field carries the software supplied data key to be used for the KeyID. Otherwise, the field is ignored.
- **KEY_FIELD_2:** If the direct key-programming command is used (TSE_SET_KEY_DIRECT), this field carries the software supplied tweak key to be used for the KeyID. Otherwise, the field is ignored.

The TSE key table is shared by all logical processors in a platform. For this reason, execution of this leaf function must gain exclusive access to the key table before updating it. The leaf function does this by acquiring a lock (implemented in the platform) and retaining that lock until the execution completes. An execution of the leaf function may fail to acquire the lock if it is already in use. In this situation, the leaf function will load EAX with failure reason 5 (DEVICE_BUSY). When this happens, the key table is not updated, and software should retry execution of PCONFIG.

NOTES

Earlier versions of this manual specified that bytes 63:6 of MKTME_KEY_PROGRAM_STRUCT were reserved and that leaf function MKTME_PROGRAM would cause a #GP if they were not all zero. This is not the case. As indicated in Table 4-16, PCONFIG ignores those bytes.

They also specified that leaf function would cause a #GP if the upper 48 bytes of each of the 64-byte key fields were not all 0. This is not the case. From each of these fields, the leaf function uses the number of bytes required by the selected encryption algorithm (e.g., 32 bytes for AES-XTS 256) and ignores the upper bytes.

They also specified that the leaf function would complete and report a failure reason in EAX if the structure specified an incorrect KeyID, and unsupported key-programming command, or an incorrect selection of an encryption algorithm. This is not the case. As indicated above (and in the Operation section), those conditions cause #GP.

Leaf Function TSE_KEY_PROGRAM_WRAPPED

PCONFIG leaf function 2 (selected by loading EAX with value 2) is used for wrapped key programming for total storage encryption (TSE). This leaf function is called TSE_KEY_PROGRAM_WRAPPED and it pertains to the TSE target, which has target identifier 2. The leaf function can be used only in 64-bit mode. It uses the RBX and RCX registers for additional input information.

Software uses this leaf function to manage the encryption key associated with a particular key identifier (KeyID). The leaf function uses control input provided in RBX. The format of that input is given in Table 4-18.

Table 4-18. TSE_KEY_PROGRAM_WRAPPED Control Input

Field	Bit Positions	Comments
KEYID	15:0	Key identifier.
Reserved	23:16	Reserved, must be zero.
ENC_ALG	39:24	Encryption algorithm.
Ignored	63:40	Not used.

A description of each of the fields in the control input is provided below:

- **KEYID:** The key identifier (KeyID) being programmed to the TSE engine. The leaf function causes a general-protection exception (#GP) if the KeyID exceeds the maximum enumerated in the TSE_MAX_KEYS field (bits 50:36) of the IA32_TSE_CAPABILITY MSR (MSR index 9F1H).
- **ENC_ALG:** The encryption algorithm selected for the KeyID. IA32_TSE_CAPABILITY[15:0] indicates which encryption algorithms are supported by the platform. The 16-bit ENC_ALG field should specify one of the algorithms indicated in IA32_TSE_CAPABILITY. The leaf function causes a #GP if ENC_ALG does not set exactly one bit or if it sets a bit whose corresponding bit is not set in IA32_TSE_CAPABILITY.

The leaf function also uses a 256-byte data structure called the **bind structure**. This structure should be the output of the PBNDKB instruction, subsequently modified by software (see below). Software provides the linear address of the structure in RCX. The format of the structure is given in Table 4-19.

Table 4-19. Bind Structure Format

Field	Offset (bytes)	Size (bytes)	Comments
MAC	0	16	MAC produced by PBNDKB of its input bind structure
Reserved	16	8	Reserved, must be zero.
IV	24	12	Initialization vector.
Reserved	36	28	Reserved, must be zero.
BTENCDATA	64	64	Encrypted data (data key and tweak key)
BTDATA	128	128	Additional control and data (not encrypted)

A description of each of the fields in TSE_BIND_STRUCT is provided below:

- **MAC:** A MAC produced by PBNDKB of its input bind structure. The PCONFIG leaf function will recompute the MAC and confirm that it matches this value.
- **IV:** The initialization vector that PBNDKB used for encryption. The PCONFIG leaf function will use this in its decryption of encrypted data and computation of the MAC.
- **BTENCDATA:** Data which had been encrypted by PBNDKB, containing the data and tweak keys to be used by TSE.
- **BTDATA:** Data that was input to PBNDKB that was output without encryption. It has the following format:
 - **USER_SUPP_CHALLENGE** (bytes 31:0): PBNDKB uses a value provided by software in its input bind structure but writes zero to this field in the output bind structure to be used by PCONFIG. Software should configure this field with the proper value before executing this PCONFIG leaf function.
 - **KEY_GENERATION_CTRL** (byte 32): PBNDKB uses this value to determine whether to generate random keys. The PCONFIG leaf function does not use this field.
 - The remaining 95 bytes are reserved and must be zero.

The leaf function uses the entire BTDATA field when it computes the MAC.

The leaf function determines a 256-bit **wrapping key** by computing an HMAC based on SHA-256 using 256-bit platform-specific key and the USER_SUPP_CHALLENGE in the BTDATA field of the TSE_BIND_STRUCT.

Using the wrapping key, the leaf function uses an AES GCM authenticated decryption function to decrypt BTENCDATA and compute a MAC. The decryption function uses the following inputs:

- The 64-byte BTENCDATA from TSE_BIND_STRUCT to be decrypted.
- The 256-bit wrapping key.
- The 96-bit IV from TSE_BIND_STRUCT.
- Additional authenticated data that is the concatenation of bytes 63:16 and bytes 255:128 of the TSE_BIND_STRUCT. These 176 bytes will comprise 8 bytes of zeroes, the 12-byte IV, 28 bytes of zeroes, and 128 bytes of BTDATA of which the upper 95 bytes are zero).
- The length of the additional authenticated data (176).

The decryption function produces a structure with a 64 bytes of decrypted data and a 16-byte MAC. The decrypted data comprises a 256-bit data key and a 256-bit tweak key.

If the MAC produced by the decryption function differs from that provided in the TSE_BIND_STRUCT, the leaf function will load EAX with failure reason 7 (UNWRAP_FAILURE). Otherwise, the leaf function will attempt to program the TSE key table for the selected KeyID with the keys contained in the decrypted data.

The TSE key table is shared by all logical processors in a platform. For this reason, execution of this leaf function must gain exclusive access to the key table before updating it. The leaf function does this by acquiring a lock (implemented in the platform) and retaining that lock until the execution completes. An execution of the leaf function may fail to acquire the lock if it is already in use. In this situation, the leaf function will load EAX with failure reason 5 (DEVICE_BUSY). When this happens, the key table is not updated, and software should retry execution of PCONFIG.

Operation

```
(* #UD if PCONFIG is not enumerated or CPL > 0 *)  
IF CPUID.07H.00H:EDX.PCONFIG[18] = 0 OR CPL > 0  
    THEN #UD; FI;
```

```
(* #GP(0) for an unsupported leaf function *)  
IF EAX > 2  
    THEN #GP(0); FI;
```

```
CASE (EAX)      (* operation based on selected leaf function *)  
  0 (MKTME_KEY_PROGRAM):  
    IF CPUID function 1BH does not enumerate support for the TME-MK target (value 1)  
        THEN #GP(0); FI;  
    (* Confirm that TME-MK is properly enabled by the IA32_TME_ACTIVATE MSR *)  
    (* The MSR must be locked, encryption enabled, and a non-zero number of KeyID bits specified *)  
    IF IA32_TME_ACTIVATE[0] = 0 OR IA32_TME_ACTIVATE[1] = 0 OR IA32_TME_ACTIVATE[35:32] = 0  
        THEN #GP(0); FI;
```

```
IF DS:RBX is not 256-byte aligned  
    THEN #GP(0); FI;
```

```
Load TMP_KEY_PROGRAM_STRUCT from 192 bytes at linear address DS:RBX;
```

```
IF TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL sets any reserved bits  
    THEN #GP(0); FI;
```

```
(* Check for a valid command *)  
IF TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.COMMAND > 3  
    THEN #GP(0); FI;
```

```
(* Check that the KEYID being operated upon is a valid KEYID *)  
IF TMP_KEY_PROGRAM_STRUCT.KEYID = 0 OR  
   TMP_KEY_PROGRAM_STRUCT.KEYID > IA32_TME_CAPABILITY.MK_TME_MAX_KEYS  
    THEN #GP(0); FI;  
k := IA32_TME_ACTIVATE.MK_TME_KEYID_BITS;  
IF TMP_KEY_PROGRAM_STRUCT.KEYID[15:k] != 0  
    THEN #GP(0); FI;  
IF not in SEAM AND IA32_TME_ACTIVATE.TDX_RESERVED_KEYID_BITS > 0  
    THEN  
        p := IA32_TME_ACTIVATE.TDX_RESERVED_KEYID_BITS;  
        IF TMP_KEY_PROGRAM_STRUCT.KEYID[k-1:k-p] != 0  
            THEN #GP(0); FI;  
FI;
```

```
(* Check that only one encryption algorithm is requested for the KeyID and it is one of the activated algorithms *)  
IF TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG does not set exactly one bit OR  
   (TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG & IA32_TME_ACTIVATE[63:48]) = 0  
    THEN #GP(0); FI;
```

```
Attempt to acquire lock to gain exclusive access to platform key table for TME-MK;  
IF attempt is unsuccessful  
    THEN (* PCONFIG failure *)  
        RFLAGS.ZF := 1;  
        RAX := DEVICE_BUSY; (* failure reason 5 *)
```

```

        GOTO EXIT;
    FI;

CASE (TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.COMMAND) OF
    0 (KEYID_SET_KEY_DIRECT):
        Update TME-MK table for TMP_KEY_PROGRAM_STRUCT.KEYID as follows:
            Encrypt with the selected key
            Use the encryption algorithm selected by TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG
            (* The number of bytes used by the next two lines depends on selected encryption algorithm *)
            DATA_KEY is TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_1
            TWEAK_KEY is TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_2
        BREAK;

    1 (KEYID_SET_KEY_RANDOM):
        Load TMP_RND_DATA_KEY with a random key using hardware RNG; (* key size depends on selected encryption algorithm *)
        IF there was insufficient entropy
            THEN (* PCONFIG failure *)
                RFLAGS.ZF := 1;
                RAX := ENTROPY_ERROR;  (* failure reason 2 *)
                Release lock on platform key table;
                GOTO EXIT;
        FI;
        Load TMP_RND_TWEAK_KEY with a random key using hardware RNG; (* key size depends on selected encryption algorithm *)
        IF there was insufficient entropy
            THEN (* PCONFIG failure *)
                RFLAGS.ZF := 1;
                RAX := ENTROPY_ERROR;  (* failure reason 2 *)
                Release lock on platform key table;
                GOTO EXIT;
        FI;
        (* Combine software-supplied entropy to the data key and tweak key *)
        (* The number of bytes used by the next two lines depends on selected encryption algorithm *)
        TMP_RND_DATA_KEY := TMP_RND_KEY XOR TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_1;
        TMP_RND_TWEAK_KEY := TMP_RND_TWEAK_KEY XOR TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_2;

        Update TME-MK table for TMP_KEY_PROGRAM_STRUCT.KEYID as follows:
            Encrypt with the selected key
            Use the encryption algorithm selected by TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG
            (* The number of bytes used by the next two lines depends on selected encryption algorithm *)
            DATA_KEY is TMP_RND_DATA_KEY
            TWEAK_KEY is TMP_RND_TWEAK_KEY
        BREAK;

    2 (KEYID_CLEAR_KEY):
        Update TME-MK table for TMP_KEY_PROGRAM_STRUCT.KEYID as follows:
            Encrypt (or not) using the current configuration for TME
            The specified encryption algorithm and key values are not used.
        BREAK;

    3 (KEYID_NO_ENCRYPT):
        Update TME-MK table for TMP_KEY_PROGRAM_STRUCT.KEYID as follows:
            Do not encrypt
            The specified encryption algorithm and key values are not used.
        BREAK;

```

ESAC;

Release lock on platform key table for TME-MK;

1 (TSE_KEY_PROGRAM):

IF CPUID function 1BH does not enumerate support for the TSE target (value 2)
THEN #GP(0); FI;

IF not in 64-bit mode
THEN #GP(0); FI;

IF RBX is not 256-byte aligned
THEN #GP(0); FI;

Load TMP_KEY_STRUCT from 192 bytes at linear address in RBX;

IF TMP_KEY_STRUCT.KEYID_CTRL sets any reserved bits
THEN #GP(0); FI;

(* Check for a valid command *)
IF TMP_KEY_STRUCT.KEYID_CTRL.COMMAND > 1
THEN #GP(0); FI;

(* Check that the KEYID being operated upon is a valid KEYID *)
IF TMP_KEY_STRUCT.KEYID > IA32_TSE_CAPABILITY.TSE_MAX_KEYS
THEN #GP(0); FI;

(* Check that only one encryption algorithm is requested for the KeyID and it is one of the activated algorithms *)
IF TMP_KEY_STRUCT.KEYID_CTRL.ENC_ALG does not set exactly one bit OR
(TMP_KEY_STRUCT.KEYID_CTRL.ENC_ALG & IA32_TSE_CAPABILITY[15:0]) = 0
THEN #GP(0); FI;

Attempt to acquire lock to gain exclusive access to platform key table for TSE;

IF attempt is unsuccessful
THEN (* PCONFIG failure *)
RFLAGS.ZF := 1;
RAX := DEVICE_BUSY; (* failure reason 5 *)
GOTO EXIT;

FI;

CASE (TMP_KEY_STRUCT.KEYID_CTRL.COMMAND) OF

0 (TSE_SET_KEY_DIRECT):

Update TSE table for TMP_KEY_STRUCT.KEYID as follows:

Encrypt with the selected key

Use the encryption algorithm selected by TMP_KEY_STRUCT.KEYID_CTRL.ENC_ALG

(* The number of bytes used by the next two lines depends on selected encryption algorithm *)

DATA_KEY is TMP_KEY_STRUCT.KEY_FIELD_1

TWEAK_KEY is TMP_KEY_STRUCT.KEY_FIELD_2

BREAK;

1 (TSE_NO_ENCRYPT):

Update TSE table for TMP_KEY_STRUCT.KEYID as follows:

Do not encrypt

The specified encryption algorithm and key values are not used.

BREAK;


```

ESAC;
Release lock on platform key table for TSE;

2 (TSE_KEY_PROGRAM_WRAPPED);
IF CPUID function 1BH does not enumerate support for the TSE target (value 2)
    THEN #GP(0); FI;

IF not in 64-bit mode OR RBX[23:16] != 0 OR RCX is not 256-byte aligned
    THEN #GP(0); FI;

(* Check that the KEYID being operated upon is a valid KEYID *)
IF RBX[15:0] > IA32_TSE_CAPABILITY.TSE_MAX_KEYS
    THEN #GP(0); FI;

(* Check that only one encryption algorithm is requested for the KeyID and it is one of the activated algorithms *)
IF RBX[39:24] does not set exactly one bit OR (RBX[39:24] & IA32_TSE_CAPABILITY[15:0]) = 0
    THEN #GP(0); FI;

Load TMP_BIND_STRUCT from 256 bytes at linear address in RCX;

(* Check TMP_BIND_STRUCT for illegal values *)
IF bytes 23:16 and bytes 63:36 of TMP_BIND_STRUCT are not all zero
    THEN #GP(0); FI;
IF TMP_BIND_STRUCT.BTDATA.KEY_GENERATION_CTRL > 1
    THEN #GP(0); FI;
IF bytes 128:33 of TMP_BIND_STRUCT.BTDATA are not all zero
    THEN #GP(0); FI;

(* Compute wrapping key *)
PLATFORM_KEY := 256-bit platform-specific key;
WRAPPING_KEY := HMAC_SHA256(PLATFORM_KEY, TMP_BIND_STRUCT.BTDATA.USER_SUPP_CHALLENGE);

(* Compose 176 bytes of additional authenticated data for use by authenticated decryption *)
AAD := Concatenation of bytes 63:16 and bytes 255:128 of TMP_BIND_STRUCT;

DECRYPT_STRUCT := AES256_GCM_DEC(TMP_BIND_STRUCT.BTENCDATA, WRAPPING_KEY, TMP_BIND_STRUCT.IV, AAD, 176);

(* Fail if MAC mismatch *)
IF TMP_BIND_STRUCT.MAC != DECRYPT_STRUCT.MAC
    THEN
        RFLAGS.ZF := 1;
        RAX := UNWRAP_FAILURE; (* failure reason 7 *)
        GOTO EXIT;
FI;

Attempt to acquire lock to gain exclusive access to platform key table for TSE;
IF attempt is unsuccessful
    THEN (* PCONFIG failure *)
        RFLAGS.ZF := 1;
        RAX := DEVICE_BUSY; (* failure reason 5 *)
        GOTO EXIT;
FI;

Update TSE table for RBX[15:0] as follows:

```

Encrypt with the selected key
 Use the encryption algorithm selected by RBX[39:24]
 (* The number of bytes used by the next two lines depends on selected encryption algorithm *)
 DATA_KEY is DECRYPT_STRUCT.DEC_DATA.KEY_FIELD_1
 TWEAK_KEY is DECRYPT_STRUCT.DEC_DATA.KEY_FIELD_2

Release lock on platform key table for TSE;

ESAC;

RAX := 0;
 RFLAGS.ZF := 0;

EXIT:
 RFLAGS.CF := 0;
 RFLAGS.PF := 0;
 RFLAGS.AF := 0;
 RFLAGS.OF := 0;
 RFLAGS.SF := 0;

Protected Mode Exceptions

#GP(0)	If input value in EAX encodes an unsupported leaf function. If a memory operand effective address is outside the relevant segment limit. MKTME_KEY_PROGRAM leaf function: If CPUID function 1BH does not enumerate support for the TME-MK target (value 1). If IA32_TME_ACTIVATE MSR is not locked. If hardware encryption and TME-MK capability are not enabled in IA32_TME_ACTIVATE MSR. If the memory operand is not 256B aligned. If any of the reserved bits in the KEYID_CTRL field of the MKTME_KEY_PROGRAM_STRUCT are set or that field indicates an unsupported KeyID, key-programming command, or encryption algorithm. TSE_KEY_PROGRAM leaf function: The TSE_KEY_PROGRAM leaf function is not supported in protected mode. TSE_KEY_PROGRAM_WRAPPED leaf function: The TSE_KEY_PROGRAM_WRAPPED leaf function is not supported in protected mode.
#PF(fault-code)	If a page fault occurs in accessing memory operands.
#UD	If any of the LOCK/REP/Operand Size/VEX prefixes are used. If current privilege level is not 0. If CPUID.07H.00H:EDX.PCONFIG[18] = 0

Real-Address Mode Exceptions

#GP	If input value in EAX encodes an unsupported leaf function. MKTME_KEY_PROGRAM leaf function: If CPUID function 1BH does not enumerate support for the TME-MK target (value 1). If IA32_TME_ACTIVATE MSR is not locked. If hardware encryption and TME-MK capability are not enabled in IA32_TME_ACTIVATE MSR. If a memory operand is not 256B aligned. If any of the reserved bits in the KEYID_CTRL field of the MKTME_KEY_PROGRAM_STRUCT are set or that field indicates an unsupported KeyID, key-programming command, or encryption algorithm.

TSE_KEY_PROGRAM leaf function:

The TSE_KEY_PROGRAM leaf function is not supported in protected mode.

TSE_KEY_PROGRAM_WRAPPED leaf function:

The TSE_KEY_PROGRAM_WRAPPED leaf function is not supported in protected mode.

#UD

If any of the LOCK/REP/Operand Size/VEX prefixes are used.

If current privilege level is not 0.

If CPUID.07H.00H:EDX.PCONFIG[18] = 0

Virtual-8086 Mode Exceptions

#UD

PCONFIG instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)

If input value in EAX encodes an unsupported leaf function.

If a memory operand is non-canonical form.

MKTME_KEY_PROGRAM leaf function:

If CPUID function 1BH does not enumerate support for the TME-MK target (value 1).

If IA32_TME_ACTIVATE MSR is not locked.

If hardware encryption and TME-MK capability are not enabled in IA32_TME_ACTIVATE MSR.

If a memory operand is not 256B aligned.

If any of the reserved bits in the KEYID_CTRL field of the MKTME_KEY_PROGRAM_STRUCT are set or that field indicates an unsupported KeyID, key-programming command, or encryption algorithm.

TSE_KEY_PROGRAM leaf function:

If CPUID function 1BH does not enumerate support for the TSE target (value 2).

If RBX is not 256-byte aligned.

If any of the reserved bits in the KEYID_CTRL field of the TMP_KEY_STRUCT are set or that field indicates an unsupported KeyID, key-programming command, or encryption algorithm.

TSE_KEY_PROGRAM_WRAPPED leaf function:

If CPUID function 1BH does not enumerate support for the TSE target (value 2).

If RCX is not 256-byte aligned.

If any of the reserved bits in RBX are set or that register indicates an unsupported KeyID or encryption algorithm.

If any of the reserved bytes in the TSE_BIND_STRUCT are set (including bytes in BTDATA).

#PF(fault-code)

If a page fault occurs in accessing memory operands.

#UD

If any of the LOCK/REP/Operand Size/VEX prefixes are used.

If the current privilege level is not 0.

If CPUID.07H.00H:EDX.PCONFIG[18] = 0.

PSRLDQ—Shift Double Quadword Right Logical

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 73 /3 ib PSRLDQ xmm1, imm8	A	V/V	SSE2	Shift xmm1 right by imm8 while shifting in 0s.
VEX.128.66.0F.WIG 73 /3 ib VPSRLDQ xmm1, xmm2, imm8	B	V/V	AVX	Shift xmm2 right by imm8 bytes while shifting in 0s.
VEX.256.66.0F.WIG 73 /3 ib VPSRLDQ ymm1, ymm2, imm8	B	V/V	AVX2	Shift ymm1 right by imm8 bytes while shifting in 0s.
EVEX.128.66.0F.WIG 73 /3 ib VPSRLDQ xmm1, xmm2/m128, imm8	C	V/V	(AVX512VL AND AVX512BW) OR AVX10.1	Shift xmm2/m128 right by imm8 bytes while shifting in 0s and store result in xmm1.
EVEX.256.66.0F.WIG 73 /3 ib VPSRLDQ ymm1, ymm2/m256, imm8	C	V/V	(AVX512VL AND AVX512BW) OR AVX10.1	Shift ymm2/m256 right by imm8 bytes while shifting in 0s and store result in ymm1.
EVEX.512.66.0F.WIG 73 /3 ib VPSRLDQ zmm1, zmm2/m512, imm8	C	V/V	AVX512BW OR AVX10.1	Shift zmm2/m512 right by imm8 bytes while shifting in 0s and store result in zmm1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:r/m (r, w)	imm8	N/A	N/A
B	N/A	VEX.vvvv (w)	ModRM:r/m (r)	imm8	N/A
C	Full Mem	EVEX.vvvv (w)	ModRM:r/m (r)	imm8	N/A

Description

Shifts the destination operand (first operand) to the right by the number of bytes specified in the count operand (second operand). The empty high-order bytes are cleared (set to all 0s). If the value specified by the count operand is greater than 15, the destination operand is set to all 0s. The count operand is an 8-bit immediate.

In 64-bit mode and not encoded with VEX/EVEX, using a REX prefix in the form of REX.R permits this instruction to access additional registers (XMM8-XMM15).

128-bit Legacy SSE version: The source and destination operands are the same. Bits (MAXVL-1:128) of the corresponding YMM destination register remain unchanged.

VEX.128 encoded version: The source and destination operands are XMM registers. Bits (MAXVL-1:128) of the destination YMM register are zeroed.

VEX.256 encoded version: The source operand is a YMM register. The destination operand is an YMM register. Bits (MAXVL-1:256) of the corresponding ZMM register are zeroed. The count operand applies to both the low and high 128-bit lanes.

EVEX encoded versions: The source operand is a ZMM/YMM/XMM register or a 512/256/128-bit memory location. The destination operand is a ZMM/YMM/XMM register. The count operand applies to each 128-bit lanes.

Note: VEX.vvvv/EVEX.vvvv encodes the destination register.

Operation

VPSRLDQ (EVEX.512 Encoded Version)

```
TEMP := COUNT
IF (TEMP > 15) THEN TEMP := 16; FI
DEST[127:0] := SRC[127:0] >> (TEMP * 8)
DEST[255:128] := SRC[255:128] >> (TEMP * 8)
DEST[383:256] := SRC[383:256] >> (TEMP * 8)
DEST[511:384] := SRC[511:384] >> (TEMP * 8)
DEST[MAXVL-1:512] := 0;
```

VPSRLDQ (VEX.256 and EVEX.256 Encoded Version)

```
TEMP := COUNT
IF (TEMP > 15) THEN TEMP := 16; FI
DEST[127:0] := SRC[127:0] >> (TEMP * 8)
DEST[255:128] := SRC[255:128] >> (TEMP * 8)
DEST[MAXVL-1:256] := 0;
```

VPSRLDQ (VEX.128 and EVEX.128 Encoded Version)

```
TEMP := COUNT
IF (TEMP > 15) THEN TEMP := 16; FI
DEST := SRC >> (TEMP * 8)
DEST[MAXVL-1:128] := 0;
```

PSRLDQ (128-bit Legacy SSE Version)

```
TEMP := COUNT
IF (TEMP > 15) THEN TEMP := 16; FI
DEST := DEST >> (TEMP * 8)
DEST[MAXVL-1:128] (Unmodified)
```

Intel C/C++ Compiler Intrinsic Equivalents

```
(V)PSRLDQ __m128i _mm_srli_si128 (__m128i a, int imm)
VPSRLDQ __m256i _mm256_srli_epi128 (__m256i, const int)
VPSRLDQ __m512i _mm512_srli_epi128 (__m512i, int)
```

Flags Affected

None.

Numeric Exceptions

None.

Other Exceptions

Non-EVEX-encoded instruction, see Table 2-24, “Type 7 Class Exception Conditions.”

EVEX-encoded instruction, see Exceptions Type E4NF.nb in Table 2-52, “Type E4NF Class Exception Conditions.”

PSHUFB—Packed Shuffle Bytes

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 38 00 /r ¹ PSHUFB mm1, mm2/m64	A	V/V	SSSE3	Shuffle bytes in mm1 according to contents of mm2/m64.
66 0F 38 00 /r PSHUFB xmm1, xmm2/m128	A	V/V	SSSE3	Shuffle bytes in xmm1 according to contents of xmm2/m128.
VEX.128.66.0F38.WIG 00 /r VPSHUFB xmm1, xmm2, xmm3/m128	B	V/V	AVX	Shuffle bytes from xmm2 into xmm1 according to contents of xmm3/m128.
VEX.256.66.0F38.WIG 00 /r VPSHUFB ymm1, ymm2, ymm3/m256	B	V/V	AVX2	Shuffle bytes from ymm2 into ymm1 according to contents of ymm3/m256.
EVEX.128.66.0F38.WIG 00 /r VPSHUFB xmm1 {k1}{z}, xmm2, xmm3/m128	C	V/V	(AVX512VL AND AVX512BW) OR AVX10.1	Shuffle bytes from xmm2 into xmm1 according to contents of xmm3/m128 under write mask k1.
EVEX.256.66.0F38.WIG 00 /r VPSHUFB ymm1 {k1}{z}, ymm2, ymm3/m256	C	V/V	(AVX512VL AND AVX512BW) OR AVX10.1	Shuffle bytes from ymm2 into ymm1 according to contents of ymm3/m256 under write mask k1.
EVEX.512.66.0F38.WIG 00 /r VPSHUFB zmm1 {k1}{z}, zmm2, zmm3/m512	C	V/V	AVX512BW OR AVX10.1	Shuffle bytes from zmm2 into zmm1 according to contents of zmm3/m512 under write mask k1.

NOTES:

- See note in Section 2.5, “Intel® AVX and Intel® SSE Instruction Exception Classification,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A, and Section 25.25.3, “Exception Conditions of Legacy SIMD Instructions Operating on MMX Registers,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (r, w)	ModRM:r/m (r)	N/A	N/A
B	N/A	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	N/A
C	Full Mem	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	N/A

Description

PSHUFB (with no VEX or EVEX prefix) performs an in-place shuffle of bytes in the destination operand (the first operand) according to the shuffle control mask in the source operand (the second operand). The instruction permutes the data in the destination operand, leaving the shuffle mask unaffected. If the most significant bit (bit[7]) of each byte of the shuffle control mask is set, then constant zero is written in the result byte. Each byte in the shuffle control mask forms an index to permute the corresponding byte in the destination operand. The value of each index is the least significant 3 bits (64-bit operation) or 4 bits (128-bit operation) of the shuffle control byte. See Figure 4-15 for an example for 64-bit operation.

The 128-bit forms of PSHUFB leave bits MAXVL–1:128 of the destination register unchanged. A 128-bit memory operand must be aligned on a 16-byte boundary or a general-protection exception (#GP) will be generated. In 64-bit mode, the REX prefix can be used to access XMM8–XMM15.

The following items apply to VPSHUFB, encoded with a VEX or EVEX prefix:

- Each of these forms uses shuffle control bytes in its second source operand to select which bytes in the first source operand to copy to the destination operand.

- These forms operate in one, two, or four 16-byte “lanes.” As with the 128-bit form of PSHUFB (above), the low 4 bits of each shuffle control byte determines which of 16 bytes in a source lane is copied to the appropriate byte in the corresponding destination lane.
- The 128-bit and 256-bit versions of these forms zero upper bits in the destination register beyond the instruction’s operand size.
- The EVEX-encoded versions update their destination conditionally with writemask k1.

Operation

PSHUFB (with 64-bit MMX operands)

```
TEMP := DEST
FOR destpos := 0 TO 7
    shufbyte := SRC.byte[destpos];
    IF shufbyte & 80H = 80H
        THEN DEST.byte[destpos] := 0;
    ELSE
        srcpos := shufbyte & 07H;
        DEST.byte[destpos] := TEMP.byte[srcpos];
FI;
```

PSHUFB (with 128-bit SSE operands)

```
TEMP := DEST;
FOR destpos := 0 TO 15
    shufbyte := SRC.byte[destpos];
    IF shufbyte & 80H = 80H
        THEN DEST.byte[destpos] := 0;
    ELSE
        srcpos := shufbyte & 0FH;
        DEST.byte[destpos] := TEMP.byte[srcpos];
FI;
```

VPSHUFB (VEX.128 encoded version)

```
FOR destpos := 0 TO 15
    shufbyte := SRC2.byte[destpos];
    IF shufbyte & 80H = 80H
        THEN DEST.byte[destpos] := 0;
    ELSE
        srcpos := shufbyte & 0FH;
        DEST.byte[destpos] := SRC1.byte[srcpos];
FI;
DEST[MAXVL-1:128] := 0;
```

VPSHUFB (VEX.256 encoded version)

```
FOR lane := 0 to 1
    FOR lanepos := 0 TO 15
        destpos := 16 * lane + lanepos;
        shufbyte := SRC2.byte[destpos];
        IF shufbyte & 80H = 80H
            THEN DEST.byte[destpos] := 0;
        ELSE
            srcpos := 16 * lane + (shufbyte & 0FH);
            DEST.byte[destpos] := SRC1.byte[srcpos];
    FI;
DEST[MAXVL-1:256] := 0;
```

VPSHUFB (EVEX encoded versions)

```
// VL is 128, 256, or 512, depending on instruction encoding
// no masking if EVEX.aaa = 0; zeroing if EVEX.z = 1
FOR lane := 0 to VL/128 - 1
  FOR lanepos := 0 TO 15
    destpos := 16 * lane + lanepos;
    IF no masking OR k[destpos] = 1      // using selected bit from k register
      THEN
        shufbyte := SRC2.byte[destpos];
        IF shufbyte & 80H = 80H
          THEN DEST.byte[destpos] := 0;
        ELSE
          srcpos := 16 * lane + (shufbyte & 0FH);
          DEST.byte[destpos] := SRC1.byte[srcpos];
      FI;
    ELSE IF zeroing                      // if not zeroing, DEST.byte[destpos] is unchanged
      THEN DEST.byte[destpos] := 0;
    FI;
  DEST[MAXVL-1:VL] := 0;
```

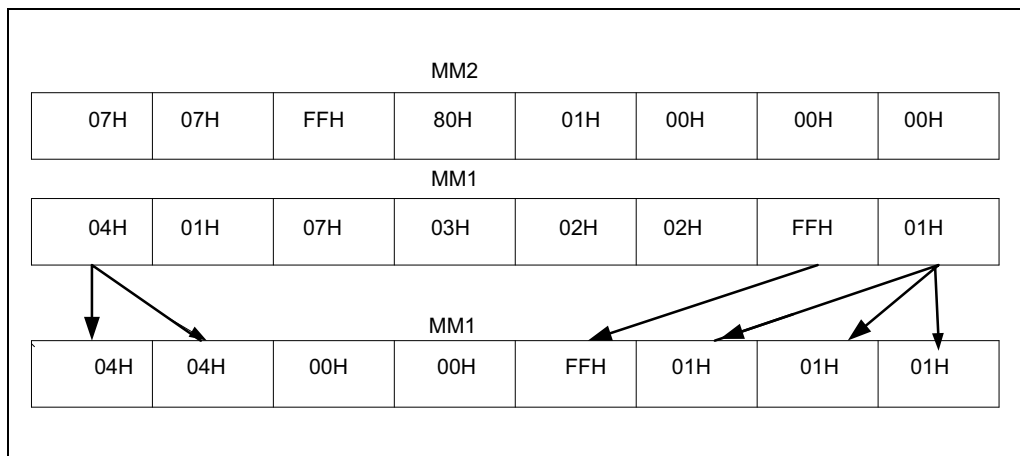


Figure 4-15. PSHUFB with 64-Bit Operands

Intel C/C++ Compiler Intrinsic Equivalent

```
VPSHUFB __m512i_mm512_shuffle_epi8(__m512i a, __m512i b);
VPSHUFB __m512i_mm512_mask_shuffle_epi8(__m512i s, __mmask64 k, __m512i a, __m512i b);
VPSHUFB __m512i_mm512_maskz_shuffle_epi8(__mmask64 k, __m512i a, __m512i b);
VPSHUFB __m256i_mm256_mask_shuffle_epi8(__m256i s, __mmask32 k, __m256i a, __m256i b);
VPSHUFB __m256i_mm256_maskz_shuffle_epi8(__mmask32 k, __m256i a, __m256i b);
VPSHUFB __m128i_mm_mask_shuffle_epi8(__m128i s, __mmask16 k, __m128i a, __m128i b);
VPSHUFB __m128i_mm_maskz_shuffle_epi8(__mmask16 k, __m128i a, __m128i b);
PSHUFB: __m64_mm_shuffle_pi8(__m64 a, __m64 b)
(V)PSHUFB: __m128i_mm_shuffle_epi8(__m128i a, __m128i b)
VPSHUFB: __m256i_mm256_shuffle_epi8(__m256i a, __m256i b)
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

Non-EVEX-encoded instruction, see Table 2-21, “Type 4 Class Exception Conditions.”

EVEX-encoded instruction, see Exceptions Type E4NF.nb in Table 2-52, “Type E4NF Class Exception Conditions.”

REP—Repeat String Operation (Prefix)

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description ¹
F3 6C	REP INS m8, DX	Z0	Valid	Valid	Input (count) bytes from port DX to (dest).
F3 6D	REP INS m16, DX	Z0	Valid	Valid	Input (count) words from port DX to (dest).
F3 6D	REP INS m32, DX	Z0	Valid	Valid	Input (count) doublewords from port DX to (dest).
F3 AC	REP LODS AL	Z0	Valid	Valid	Load (count) bytes from (src) to AL.
F3 AD	REP LODS AX	Z0	Valid	Valid	Load (count) words from (src) to AX.
F3 AD	REP LODS EAX	Z0	Valid	Valid	Load (count) doublewords from (src) to EAX.
F3 REX.W AD	REP LODS RAX	Z0	Valid	N.E.	Load (count) quadwords from (src) to RAX.
F3 A4	REP MOVSB m8, m8	Z0	Valid	Valid	Move (count) bytes from (src) to (dest).
F3 A5	REP MOVSW m16, m16	Z0	Valid	Valid	Move (count) words from (src) to (dest).
F3 A5	REP MOVSD m32, m32	Z0	Valid	Valid	Move (count) doublewords from (src) to (dest).
F3 REX.W A5	REP MOVSD m64, m64	Z0	Valid	N.E.	Move (count) quadwords from (src) to (dest).
F3 6E	REP OUTSB DX, m8	Z0	Valid	Valid	Output (count) bytes from (src) to port DX.
F3 6F	REP OUTSW DX, m16	Z0	Valid	Valid	Output (count) words from (src) to port DX.
F3 6F	REP OUTSD DX, m32	Z0	Valid	Valid	Output (count) doublewords from (src) to port DX.
F3 AA	REP STOSB m8	Z0	Valid	Valid	Fill (count) bytes at (dest) with AL.
F3 AB	REP STOSW m16	Z0	Valid	Valid	Fill (count) words at (dest) with AX.
F3 AB	REP STOSD m32	Z0	Valid	Valid	Fill (count) doublewords at (dest) with EAX.
F3 REX.W AB	REP STOSD m64	Z0	Valid	N.E.	Fill (count) quadwords at (dest) with RAX.

NOTES:

1. See Description section for details on (count), (src), and (dest).

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

Repeats a string instruction the number of times specified in the count register. The count register is CX, ECX, or RCX, depending on the instruction's address size. The REP (repeat) mnemonic is a prefix that can be added to the INS, OUTS, MOVSB, LODSB, and STOSB instructions.

The REP prefixes apply only to one string instruction at a time. To repeat a block of instructions, use the LOOP instruction or another looping construct. The REP prefixes causes the associated instruction to be repeated until the count in register is decremented to 0.

Each of the string instructions uses a source address, a destination address, or both. The source address is DS:SI, DS:ESI, or DS:RSI, depending on the instruction's address size; the DS segment may be overridden by an instruction prefix. The destination address is ES:DI, ES:EDI, or ES:RDI, depending on the instruction's address size; the ES segment may not be overridden. (Note that, in 64-bit mode, the base addresses of the CS, DS, ES, and SS segments are treated as zero.)

Similarly, the size of the count register is the instruction's address size. Thus, the default count register in 64-bit mode is RCX; REX.W has no effect on the address size and the count register. If 67H is used to override the default address size, the size of the count register is also overridden.

A repeating string operation can be suspended by an exception or interrupt. When this happens, the state of the registers is preserved to allow the string operation to be resumed upon a return from the exception or interrupt handler. The source and destination registers point to the next string elements to be operated on, the EIP register points to the string instruction, and the ECX register has the value it held following the last successful iteration of the instruction. This mechanism allows long string operations to proceed without affecting the interrupt response time of the system.

Use the REP INS and REP OUTS instructions with caution. Not all I/O ports can handle the rate at which these instructions execute. Note that a REP STOS instruction is the fastest way to initialize a large block of memory.

REP INS may read from the I/O port without writing to the memory location if an exception or VM exit occurs due to the write (e.g., #PF). If this would be problematic, for example because the I/O port read has side-effects, software should ensure the write to the memory location does not cause an exception or VM exit.

Operation

```
IF AddressSize = 16
  THEN
    Use CX for CountReg;
    Implicit Source/Dest operand for memory use of SI/DI;
ELSE IF AddressSize = 64
  THEN Use RCX for CountReg;
    Implicit Source/Dest operand for memory use of RSI/RDI;
ELSE
  Use ECX for CountReg;
  Implicit Source/Dest operand for memory use of ESI/EDI;
FI;
WHILE CountReg ≠ 0
  DO
    Service pending interrupts (if any);
    Execute associated string instruction;
    CountReg := (CountReg - 1);
  OD;
```

Flags Affected

None.

Exceptions (All Operating Modes)

Exceptions may be generated by an instruction associated with the prefix.

RET—Return From Procedure

Opcode*	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
C3	RET	Z0	Valid	Valid	Near return to calling procedure.
CB	RET	Z0	Valid	Valid	Far return to calling procedure.
C2 iw	RET imm16	I	Valid	Valid	Near return to calling procedure and pop imm16 bytes from stack.
CA iw	RET imm16	I	Valid	Valid	Far return to calling procedure and pop imm16 bytes from stack.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A
I	imm16	N/A	N/A	N/A

Description

Transfers program control to a return address located on the top of the stack. The address is usually placed on the stack by a CALL instruction, and the return is made to the instruction that follows the CALL instruction.

The optional source operand specifies the number of stack bytes to be released after the return address is popped; the default is none. This operand can be used to release parameters from the stack that were passed to the called procedure and are no longer needed. It must be used when the CALL instruction used to switch to a new procedure uses a call gate with a non-zero word count to access the new procedure. Here, the source operand for the RET instruction must specify the same number of bytes as is specified in the word count field of the call gate.

The RET instruction can be used to execute three different types of returns:

- **Near return** — A return to a calling procedure within the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intrasegment return.
- **Far return** — A return to a calling procedure located in a different segment than the current code segment, sometimes referred to as an intersegment return.
- **Inter-privilege-level far return** — A far return to a different privilege level than that of the currently executing program or procedure.

The inter-privilege-level return type can only be executed in protected mode. See the section titled “Calling Procedures Using Call and RET” in Chapter 6 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for detailed information on near, far, and inter-privilege-level returns.

When executing a near return, the processor pops the return instruction pointer (offset) from the top of the stack into the EIP register and begins program execution at the new instruction pointer. The CS register is unchanged.

When executing a far return, the processor pops the return instruction pointer from the top of the stack into the EIP register, then pops the segment selector from the top of the stack into the CS register. The processor then begins program execution in the new code segment at the new instruction pointer.

The mechanics of an inter-privilege-level far return are similar to an intersegment return, except that the processor examines the privilege levels and access rights of the code and stack segments being returned to determine if the control transfer is allowed to be made. The DS, ES, FS, and GS segment registers are cleared by the RET instruction during an inter-privilege-level return if they refer to segments that are not allowed to be accessed at the new privilege level. Since a stack switch also occurs on an inter-privilege level return, the ESP and SS registers are loaded from the stack.

If parameters are passed to the called procedure during an inter-privilege level call, the optional source operand must be used with the RET instruction to release the parameters on the return. Here, the parameters are released both from the called procedure’s stack and the calling procedure’s stack (that is, the stack being returned to).

In 64-bit mode, the default operation size of this instruction is the stack-address size, i.e., 64 bits. This applies to near returns, not far returns; the default operation size of far returns is 32 bits.

Refer to Chapter 6, “Procedure Calls, Interrupts, and Exceptions,” and Chapter 18, “Control-flow Enforcement Technology (CET),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for CET details.

When FRED transitions are enabled, an execution of far RET that would change CPL causes a general-protection exception, as does an execution of far RET that would enter compatibility mode when CPL is 0.

Instruction ordering. Instructions following a far return may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the far return have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Unlike near indirect CALL and near indirect JMP, the processor will not speculatively execute the next sequential instruction after a near RET unless that instruction is also the target of a jump or is a target in a branch predictor.

Operation

(* Near return *)

IF instruction = near return

THEN;

IF OperandSize = 32

THEN

IF top 4 bytes of stack not within stack limits

THEN #SS(0); FI;

EIP := Pop();

IF ShadowStackEnabled(CPL)

tempSsEIP = ShadowStackPop4B();

IF EIP != TempSsEIP

THEN #CP(NEAR_RET); FI;

FI;

ELSE

IF OperandSize = 64

THEN

IF top 8 bytes of stack not within stack limits

THEN #SS(0); FI;

RIP := Pop();

IF ShadowStackEnabled(CPL)

tempSsEIP = ShadowStackPop8B();

IF RIP != tempSsEIP

THEN #CP(NEAR_RET); FI;

FI;

ELSE (* OperandSize = 16 *)

IF top 2 bytes of stack not within stack limits

THEN #SS(0); FI;

tempEIP := Pop();

tempEIP := tempEIP AND 0000FFFFH;

IF tempEIP not within code segment limits

THEN #GP(0); FI;

EIP := tempEIP;

IF ShadowStackEnabled(CPL)

tempSsEip = ShadowStackPop4B();

IF EIP != tempSsEIP

THEN #CP(NEAR_RET); FI;

FI;

FI;

FI;

IF instruction has immediate operand

```

    THEN (* Release parameters from stack *)
        IF StackAddressSize = 32
            THEN
                ESP := ESP + SRC;
            ELSE
                IF StackAddressSize = 64
                    THEN
                        RSP := RSP + SRC;
                    ELSE (* StackAddressSize = 16 *)
                        SP := SP + SRC;
                FI;
            FI;
        FI;
    FI;

(* Real-address mode or virtual-8086 mode *)
IF ((PE = 0) or (PE = 1 AND VM = 1)) and instruction = far return
    THEN
        IF OperandSize = 32
            THEN
                IF top 8 bytes of stack not within stack limits
                    THEN #SS(0); FI;
                EIP := Pop();
                CS := Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            ELSE (* OperandSize = 16 *)
                IF top 4 bytes of stack not within stack limits
                    THEN #SS(0); FI;
                tempEIP := Pop();
                tempEIP := tempEIP AND 0000FFFFH;
                IF tempEIP not within code segment limits
                    THEN #GP(0); FI;
                EIP := tempEIP;
                CS := Pop(); (* 16-bit pop *)
            FI;
        IF instruction has immediate operand
            THEN (* Release parameters from stack *)
                SP := SP + (SRC AND FFFFH);
            FI;
    FI;

(* Protected mode, not virtual-8086 mode *)
IF (PE = 1 and VM = 0 and IA32_EFER.LMA = 0) and instruction = far return
    THEN
        IF OperandSize = 32
            THEN
                IF second doubleword on stack is not within stack limits
                    THEN #SS(0); FI;
            ELSE (* OperandSize = 16 *)
                IF second word on stack is not within stack limits
                    THEN #SS(0); FI;
            FI;
        IF return code segment selector is NULL
            THEN #GP(0); FI;
        IF return code segment selector addresses descriptor beyond descriptor table limit

```

```

        THEN #GP(selector); FI;
    Obtain descriptor to which return code segment selector points from descriptor table;
    IF return code segment descriptor is not a code segment
        THEN #GP(selector); FI;
    IF return code segment selector RPL < CPL
        THEN #GP(selector); FI;
    IF return code segment descriptor is conforming and return code segment DPL > return code segment selector RPL
        THEN #GP(selector); FI;
    IF return code segment descriptor is non-conforming and return code segment DPL ≠ return code segment selector RPL
        THEN #GP(selector); FI;
    IF return code segment descriptor is not present
        THEN #NP(selector); FI;
    IF return code segment selector RPL > CPL
        THEN GOTO RETURN-TO-OUTER-PRIVILEGE-LEVEL;
        ELSE GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL;
    FI;
FI;

RETURN-TO-SAME-PRIVILEGE-LEVEL:
    IF the return instruction pointer is not within the return code segment limit
        THEN #GP(0); FI;
    IF OperandSize = 32
        THEN
            EIP := Pop();
            CS := Pop(); (* 32-bit pop, high-order 16 bits discarded *)
        ELSE (* OperandSize = 16 *)
            EIP := Pop();
            EIP := EIP AND 0000FFFFH;
            CS := Pop(); (* 16-bit pop *)
        FI;
    IF instruction has immediate operand
        THEN (* Release parameters from stack *)
            IF StackAddressSize = 32
                THEN
                    ESP := ESP + SRC;
                ELSE (* StackAddressSize = 16 *)
                    SP := SP + SRC;
            FI;
        FI;
    IF ShadowStackEnabled(CPL)
        (* SSP must be 8 byte aligned *)
        IF SSP AND 0x7 ≠ 0
            THEN #CP(FAR-RET/IRET); FI;
        tempSsCS = shadow_stack_load 8 bytes from SSP+16;
        tempSsLIP = shadow_stack_load 8 bytes from SSP+8;
        prevSSP = shadow_stack_load 8 bytes from SSP;
        SSP = SSP + 24;
        (* do a 64 bit-compare to check if any bits beyond bit 15 are set *)
        tempCS = CS; (* zero pad to 64 bit *)
        IF tempCS ≠ tempSsCS
            THEN #CP(FAR-RET/IRET); FI;
        (* do a 64 bit-compare; pad CSBASE+RIP with 0 for 32 bit LIP*)
        IF CSBASE + RIP ≠ tempSsLIP
            THEN #CP(FAR-RET/IRET); FI;

```

```

(* prevSSP must be 4 byte aligned *)
IF prevSSP AND 0x3 != 0
    THEN #CP(FAR-RET/IRET); FI;
(* In legacy mode SSP must be in low 4GB *)
IF prevSSP[63:32] != 0
    THEN #GP(0); FI;
SSP := prevSSP
FI;

```

RETURN-TO-OUTER-PRIVILEGE-LEVEL:

```

IF top (16 + SRC) bytes of stack are not within stack limits (OperandSize = 32)
or top (8 + SRC) bytes of stack are not within stack limits (OperandSize = 16)
    THEN #SS(0); FI;
Read return segment selector;
IF stack segment selector is NULL
    THEN #GP(0); FI;
IF return stack segment selector index is not within its descriptor table limits
    THEN #GP(selector); FI;
Read segment descriptor pointed to by return segment selector;
IF stack segment selector RPL ≠ RPL of the return code segment selector
or stack segment is not a writable data segment
or stack segment descriptor DPL ≠ RPL of the return code segment selector
    THEN #GP(selector); FI;
IF stack segment not present
    THEN #SS(StackSegmentSelector); FI;
IF the return instruction pointer is not within the return code segment limit
    THEN #GP(0); FI;
IF OperandSize = 32
    THEN
        EIP := Pop();
        CS := Pop(); (* 32-bit pop, high-order 16 bits discarded; segment descriptor loaded *)
        CS(RPL) := ReturnCodeSegmentSelector(RPL);
        IF instruction has immediate operand
            THEN (* Release parameters from called procedure's stack *)
                IF StackAddressSize = 32
                    THEN
                        ESP := ESP + SRC;
                    ELSE (* StackAddressSize = 16 *)
                        SP := SP + SRC;
                FI;
            FI;
        tempESP := Pop();
        tempSS := Pop(); (* 32-bit pop, high-order 16 bits discarded; seg. descriptor loaded *)
    ELSE (* OperandSize = 16 *)
        EIP := Pop();
        EIP := EIP AND 0000FFFFH;
        CS := Pop(); (* 16-bit pop; segment descriptor loaded *)
        CS(RPL) := ReturnCodeSegmentSelector(RPL);
        IF instruction has immediate operand
            THEN (* Release parameters from called procedure's stack *)
                IF StackAddressSize = 32
                    THEN
                        ESP := ESP + SRC;
                    ELSE (* StackAddressSize = 16 *)

```



```

        SP := SP + SRC;
    FI;
    FI;
    tempESP := Pop();
    tempSS := Pop(); (* 16-bit pop; segment descriptor loaded *)
FI;
IF ShadowStackEnabled(CPL)
    (* check if 8 byte aligned *)
    IF SSP AND 0x7 != 0
        THEN #CP(FAR-RET/IRET); FI;
    IF ReturnCodeSegmentSelector(RPL) != 3
        THEN
            tempSsCS = shadow_stack_load 8 bytes from SSP+16;
            tempSsLIP = shadow_stack_load 8 bytes from SSP+8;
            tempSSP = shadow_stack_load 8 bytes from SSP;
            SSP = SSP + 24;
            (* Do 64 bit compare to detect bits beyond 15 being set *)
            tempCS = CS; (* zero extended to 64 bit *)
            IF tempCS != tempSsCS
                THEN #CP(FAR-RET/IRET); FI;
            (* Do 64 bit compare; pad CSBASE+RIP with 0 for 32 bit LA *)
            IF CSBASE + RIP != tempSsLIP
                THEN #CP(FAR-RET/IRET); FI;
            (* check if 4 byte aligned *)
            IF tempSSP AND 0x3 != 0
                THEN #CP(FAR-RET/IRET); FI;
FI;
FI;
tempOldCPL = CPL;

CPL := ReturnCodeSegmentSelector(RPL);
ESP := tempESP;
SS := tempSS;
tempOldSSP = SSP;
IF ShadowStackEnabled(CPL)
    IF CPL = 3
        THEN tempSSP := IA32_PL3_SSP; FI;
    IF tempSSP[63:32] != 0
        THEN #GP(0); FI;
    SSP := tempSSP
FI;
(* Now past all faulting points; safe to free the token. The token free is done using the old SSP
* and using a supervisor override as old CPL was a supervisor privilege level *)
IF ShadowStackEnabled(tempOldCPL)
    expected_token_value = tempOldSSP | BUSY_BIT (* busy bit - bit position 0 - must be set *)
    new_token_value = tempOldSSP (* clear the busy bit *)
    shadow_stack_lock_cmpxchg8b(tempOldSSP, new_token_value, expected_token_value)
FI;
FI;

FOR each SegReg in (ES, FS, GS, and DS)
    DO
        tempDesc := descriptor cache for SegReg (* hidden part of segment register *)
        IF (SegmentSelector == NULL) OR (tempDesc(DPL) < CPL AND tempDesc(Type) is (data or non-conforming code)))

```

```

        THEN (* Segment register invalid *)
            SegmentSelector := 0; (*Segment selector becomes null*)
    FI;
OD;

IF instruction has immediate operand
    THEN (* Release parameters from calling procedure's stack *)
        IF StackAddressSize = 32
            THEN
                ESP := ESP + SRC;
            ELSE (* StackAddressSize = 16 *)
                SP := SP + SRC;
            FI;
    FI;

(* IA-32e Mode *)
IF (PE = 1 and VM = 0 and IA32_EFER.LMA = 1) and instruction = far return
    THEN
        IF OperandSize = 32
            THEN
                IF second doubleword on stack is not within stack limits
                    THEN #SS(0); FI;
                IF first or second doubleword on stack is not in canonical space
                    THEN #SS(0); FI;
            ELSE
                IF OperandSize = 16
                    THEN
                        IF second word on stack is not within stack limits
                            THEN #SS(0); FI;
                        IF first or second word on stack is not in canonical space
                            THEN #SS(0); FI;
                    ELSE (* OperandSize = 64 *)
                        IF first or second quadword on stack is not in canonical space
                            THEN #SS(0); FI;
                    FI;
            FI;
        FI;
    IF return code segment selector is NULL
        THEN GP(0); FI;
    IF return code segment selector addresses descriptor beyond descriptor table limit
        THEN GP(selector); FI;
    IF return code segment selector addresses descriptor in non-canonical space
        THEN GP(selector); FI;
    Obtain descriptor to which return code segment selector points from descriptor table;
    IF return code segment descriptor is not a code segment
        THEN #GP(selector); FI;
    IF return code segment descriptor has L-bit = 1 and D-bit = 1
        THEN #GP(selector); FI;
    IF return code segment selector RPL < CPL or (CR4.FRED = 1 and return code segment selector RPL > CPL)
        THEN #GP(selector); FI;
    IF return code segment descriptor is conforming and return code segment DPL > return code segment selector RPL
        THEN #GP(selector); FI;
    IF return code segment descriptor is non-conforming and return code segment DPL ≠ return code segment selector RPL
        THEN #GP(selector); FI;
    IF CR4.FRED = 1 and CPL = 0 and L-bit is 0 in return code segment descriptor

```

```

        THEN #GP(selector); FI;
    IF return code segment descriptor is not present
        THEN #NP(selector); FI;
    IF return code segment selector RPL > CPL
        THEN GOTO IA-32E-MODE-RETURN-TO-OUTER-PRIVILEGE-LEVEL;
        ELSE GOTO IA-32E-MODE-RETURN-TO-SAME-PRIVILEGE-LEVEL;
    FI;
FI;

IA-32E-MODE-RETURN-TO-SAME-PRIVILEGE-LEVEL:
IF the return instruction pointer is not within the return code segment limit
    THEN #GP(0); FI;
IF the return instruction pointer is not within canonical address space
    THEN #GP(0); FI;
IF OperandSize = 32
    THEN
        EIP := Pop();
        CS := Pop(); (* 32-bit pop, high-order 16 bits discarded *)
    ELSE
        IF OperandSize = 16
            THEN
                EIP := Pop();
                EIP := EIP AND 0000FFFFH;
                CS := Pop(); (* 16-bit pop *)
            ELSE (* OperandSize = 64 *)
                RIP := Pop();
                CS := Pop(); (* 64-bit pop, high-order 48 bits discarded *)
        FI;
    FI;
FI;

IF instruction has immediate operand
    THEN (* Release parameters from stack *)
        IF StackAddressSize = 32
            THEN
                ESP := ESP + SRC;
            ELSE
                IF StackAddressSize = 16
                    THEN
                        SP := SP + SRC;
                    ELSE (* StackAddressSize = 64 *)
                        RSP := RSP + SRC;
                FI;
            FI;
        FI;
    FI;
FI;

IF ShadowStackEnabled(CPL)
    IF SSP AND 0x7 != 0 (* check if aligned to 8 bytes *)
        THEN #CP(FAR-RET/IRET); FI;
    tempSsCS = shadow_stack_load 8 bytes from SSP+16;
    tempSsLIP = shadow_stack_load 8 bytes from SSP+8;
    tempSSP = shadow_stack_load 8 bytes from SSP;
    SSP = SSP + 24;
    tempCS = CS; (* zero padded to 64 bit *)
    IF tempCS != tempSsCS (* 64 bit compare; CS zero padded to 64 bits *)
        THEN #CP(FAR-RET/IRET); FI;
    IF CSBASE + RIP != tempSsLIP (* 64 bit compare *)

```

```

    THEN #CP(FAR-RET/IRET); FI;
IF tempSSP AND 0x3 != 0 (* check if aligned to 4 bytes *)
    THEN #CP(FAR-RET/IRET); FI;
IF (CS.L = 0 AND tempSSP[63:32] != 0) OR
    (CS.L = 1 AND tempSSP is not canonical relative to the current paging mode)
    THEN #GP(0); FI;
SSP := tempSSP
FI;

IA-32E-MODE-RETURN-TO-OUTER-PRIVILEGE-LEVEL:
IF top (16 + SRC) bytes of stack are not within stack limits (OperandSize = 32)
or top (8 + SRC) bytes of stack are not within stack limits (OperandSize = 16)
    THEN #SS(0); FI;
IF top (16 + SRC) bytes of stack are not in canonical address space (OperandSize = 32)
or top (8 + SRC) bytes of stack are not in canonical address space (OperandSize = 16)
or top (32 + SRC) bytes of stack are not in canonical address space (OperandSize = 64)
    THEN #SS(0); FI;
Read return stack segment selector;
IF stack segment selector is NULL
    THEN
        IF new CS descriptor L-bit = 0
            THEN #GP(selector);
        IF stack segment selector RPL = 3
            THEN #GP(selector);
FI;
IF return stack segment descriptor is not within descriptor table limits
    THEN #GP(selector); FI;
IF return stack segment descriptor is in non-canonical address space
    THEN #GP(selector); FI;
Read segment descriptor pointed to by return segment selector;
IF stack segment selector RPL ≠ RPL of the return code segment selector
or stack segment is not a writable data segment
or stack segment descriptor DPL ≠ RPL of the return code segment selector
    THEN #GP(selector); FI;
IF stack segment not present
    THEN #SS(StackSegmentSelector); FI;
IF the return instruction pointer is not within the return code segment limit
    THEN #GP(0); FI;
IF the return instruction pointer is not within canonical address space
    THEN #GP(0); FI;
IF OperandSize = 32
    THEN
        EIP := Pop();
        CS := Pop(); (* 32-bit pop, high-order 16 bits discarded, segment descriptor loaded *)
        CS(RPL) := ReturnCodeSegmentSelector(RPL);
        IF instruction has immediate operand
            THEN (* Release parameters from called procedure's stack *)
                IF StackAddressSize = 32
                    THEN
                        ESP := ESP + SRC;
                    ELSE
                        IF StackAddressSize = 16
                            THEN
                                SP := SP + SRC;

```

```

        ELSE (* StackAddressSize = 64 *)
            RSP := RSP + SRC;
        FI;
    FI;
FI;
tempESP := Pop();
tempSS := Pop(); (* 32-bit pop, high-order 16 bits discarded, segment descriptor loaded *)
ELSE
    IF OperandSize = 16
        THEN
            EIP := Pop();
            EIP := EIP AND 0000FFFFH;
            CS := Pop(); (* 16-bit pop; segment descriptor loaded *)
            CS(RPL) := ReturnCodeSegmentSelector(RPL);
            IF instruction has immediate operand
                THEN (* Release parameters from called procedure's stack *)
                    IF StackAddressSize = 32
                        THEN
                            ESP := ESP + SRC;
                        ELSE
                            IF StackAddressSize = 16
                                THEN
                                    SP := SP + SRC;
                                ELSE (* StackAddressSize = 64 *)
                                    RSP := RSP + SRC;
                                FI;
                            FI;
                        FI;
                    FI;
                tempESP := Pop();
                tempSS := Pop(); (* 16-bit pop; segment descriptor loaded *)
            ELSE (* OperandSize = 64 *)
                RIP := Pop();
                CS := Pop(); (* 64-bit pop; high-order 48 bits discarded; seg. descriptor loaded *)
                CS(RPL) := ReturnCodeSegmentSelector(RPL);
                IF instruction has immediate operand
                    THEN (* Release parameters from called procedure's stack *)
                        RSP := RSP + SRC;
                    FI;
                tempESP := Pop();
                tempSS := Pop(); (* 64-bit pop; high-order 48 bits discarded; seg. desc. loaded *)
            FI;
        FI;
    FI;

IF ShadowStackEnabled(CPL)
    (* check if 8 byte aligned *)
    IF SSP AND 0x7 != 0
        THEN #CP(FAR-RET/IRET); FI;
    IF ReturnCodeSegmentSelector(RPL) != 3
        THEN
            tempSsCS = shadow_stack_load 8 bytes from SSP+16;
            tempSsLIP = shadow_stack_load 8 bytes from SSP+8;
            tempSSP = shadow_stack_load 8 bytes from SSP;
            SSP = SSP + 24;
            (* Do 64 bit compare to detect bits beyond 15 being set *)

```

```

    tempCS = CS; (* zero padded to 64 bit *)
    IF tempCS != tempSsCS
        THEN #CP(FAR-RET/IRET); FI;
    (* Do 64 bit compare; pad CSBASE+RIP with 0 for 32 bit LIP *)
    IF CSBASE + RIP != tempSsLIP
        THEN #CP(FAR-RET/IRET); FI;
    (* check if 4 byte aligned *)
    IF tempSSP AND 0x3 != 0
        THEN #CP(FAR-RET/IRET); FI;
FI;
FI;
tempOldCPL = CPL;
CPL := ReturnCodeSegmentSelector(RPL);
ESP := tempESP;
SS := tempSS;
tempOldSSP = SSP;
IF ShadowStackEnabled(CPL)
    IF CPL = 3
        THEN tempSSP := IA32_PL3_SSP; FI;
    IF (CS.L = 0 AND tempSSP[63:32] != 0) OR
        (CS.L = 1 AND tempSSP is not canonical relative to the current paging mode)
        THEN #GP(0); FI;
    SSP := tempSSP
FI;
(* Now past all faulting points; safe to free the token. The token free is done using the old SSP
* and using a supervisor override as old CPL was a supervisor privilege level *)
IF ShadowStackEnabled(tempOldCPL)
    expected_token_value = tempOldSSP | BUSY_BIT      (* busy bit - bit position 0 - must be set *)
    new_token_value = tempOldSSP                    (* clear the busy bit *)
    shadow_stack_lock_cmpxchg8b(tempOldSSP, new_token_value, expected_token_value)
FI;

FOR each of segment register (ES, FS, GS, and DS)
    DO
        IF segment register points to data or non-conforming code segment
        and CPL > segment descriptor DPL; (* DPL in hidden part of segment register *)
            THEN SegmentSelector := 0; (* SegmentSelector invalid *)
        FI;
    OD;

IF instruction has immediate operand
    THEN (* Release parameters from calling procedure's stack *)
        IF StackAddressSize = 32
            THEN
                ESP := ESP + SRC;
            ELSE
                IF StackAddressSize = 16
                    THEN
                        SP := SP + SRC;
                    ELSE (* StackAddressSize = 64 *)
                        RSP := RSP + SRC;
                FI;
            FI;
        FI;
FI;

```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If the return code or stack segment selector is NULL. If the return instruction pointer is not within the return code segment limit. If returning to 32-bit or compatibility mode and the previous SSP from shadow stack (when returning to CPL <3) or from IA32_PL3_SSP (returning to CPL 3) is beyond 4GB.
#GP(selector)	If the RPL of the return code segment selector is less than the CPL. If the return code or stack segment selector index is not within its descriptor table limits. If the return code segment descriptor does not indicate a code segment. If the return code segment is non-conforming and the segment selector's DPL is not equal to the RPL of the code segment's segment selector If the return code segment is conforming and the segment selector's DPL greater than the RPL of the code segment's segment selector If the stack segment is not a writable data segment. If the stack segment selector RPL is not equal to the RPL of the return code segment selector. If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector.
#SS(0)	If the top bytes of stack are not within stack limits. If the return stack segment is not present.
#NP(selector)	If the return code segment is not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory access occurs when the CPL is 3 and alignment checking is enabled.
#CP(Far-RET/IRET)	If the previous SSP from shadow stack (when returning to CPL <3) or from IA32_PL3_SSP (returning to CPL 3) is not 4 byte aligned. If return instruction pointer from stack and shadow stack do not match.

Real-Address Mode Exceptions

#GP	If the return instruction pointer is not within the return code segment limit
#SS	If the top bytes of stack are not within stack limits.

Virtual-8086 Mode Exceptions

#GP(0)	If the return instruction pointer is not within the return code segment limit
#SS(0)	If the top bytes of stack are not within stack limits.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory access occurs when alignment checking is enabled.

Compatibility Mode Exceptions

Same as 64-bit mode exceptions.

64-Bit Mode Exceptions

#GP(0)	If the return instruction pointer is non-canonical. If the return instruction pointer is not within the return code segment limit. If the stack segment selector is NULL going back to compatibility mode. If the stack segment selector is NULL going back to CPL3 64-bit mode. If a NULL stack segment selector RPL is not equal to CPL going back to non-CPL3 64-bit mode. If the return code segment selector is NULL. If returning to 32-bit or compatibility mode and the previous SSP from shadow stack (when returning to CPL <3) or from IA32_PL3_SSP (returning to CPL 3) is beyond 4GB.
#GP(selector)	If the proposed segment descriptor for a code segment does not indicate it is a code segment. If the proposed new code segment descriptor has both the D-bit and L-bit set. If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment selector. If CPL is greater than the RPL of the code segment selector. If FRED transitions are enabled and CPL is less than the RPL of the code segment selector. If FRED transitions are enabled and CPL = 0 and far RET would enter compatibility mode. If the DPL of a conforming-code segment is greater than the return code segment selector RPL. If a segment selector index is outside its descriptor table limits. If a segment descriptor memory address is non-canonical. If the stack segment is not a writable data segment. If the stack segment descriptor DPL is not equal to the RPL of the return code segment selector. If the stack segment selector RPL is not equal to the RPL of the return code segment selector.
#SS(0)	If an attempt to pop a value off the stack violates the SS limit. If an attempt to pop a value off the stack causes a non-canonical address to be referenced.
#NP(selector)	If the return code or stack segment is not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#CP(Far-RET/IRET)	If the previous SSP from shadow stack (when returning to CPL <3) or from IA32_PL3_SSP (returning to CPL 3) is not 4 byte aligned. If return instruction pointer from stack and shadow stack do not match.

SAVEPREVSSP—Save Previous Shadow Stack Pointer

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 01 EA SAVEPREVSSP	Z0	V/V	CET_SS	Save a restore-shadow-stack token on previous shadow stack.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

Push a restore-shadow-stack token on the previous shadow stack at the next 8 byte aligned boundary. The previous SSP is obtained from the previous-ssp token at the top of the current shadow stack.

Operation

IF CPL = 3

IF (CR4.CET & IA32_U_CET.SH_STK_EN) = 0
THEN #UD; FI;

ELSE

IF (CR4.CET & IA32_S_CET.SH_STK_EN) = 0
THEN #UD; FI;

FI;

IF SSP not aligned to 8 bytes

THEN #GP(0); FI;

(* Pop the “previous-ssp” token from current shadow stack *)

previous_ssp_token = ShadowStackPop8B(SSP)

(* If the CF flag indicates there was a alignment hole on current shadow stack then pop that alignment hole *)

(* Note that the alignment hole must be zero and can be present only when in legacy/compatibility mode *)

IF RFLAGS.CF == 1 AND (IA32_EFER.LMA AND CS.L)

#GP(0)

FI;

IF RFLAGS.CF == 1

must_be_zero = ShadowStackPop4B(SSP)

IF must_be_zero != 0 THEN #GP(0)

FI;

(* Previous SSP token must have the bit 1 set *)

IF ((previous_ssp_token & 0x02) == 0)

THEN #GP(0); (* bit 1 was 0 *)

IF ((IA32_EFER.LMA AND CS.L) = 0 AND previous_ssp_token [63:32] != 0)

THEN #GP(0); FI; (* If compatibility/legacy mode and SSP not in 4G *)

(* Save Prev SSP from previous_ssp_token to the old shadow stack at next 8 byte aligned address *)

old_SSP = previous_ssp_token & ~0x03

temp := (old_SSP | (IA32_EFER.LMA & CS.L));

Shadow_stack_store 4 bytes of 0 to (old_SSP - 4)

old_SSP := old_SSP & ~0x07;

Shadow_stack_store 8 bytes of temp to (old_SSP - 8)

Flags Affected

None.

C/C++ Compiler Intrinsic Equivalent

SAVEPREVSSP void _saveprevssp(void);

Protected Mode Exceptions

#UD	If the LOCK prefix is used. If CR4.CET = 0. If CPL = 3 and IA32_U_CET.SH_STK_EN = 0. If CPL < 3 and IA32_S_CET.SH_STK_EN = 0.
#GP(0)	If SSP not 8 byte aligned. If alignment hole on shadow stack is not 0. If bit 1 of the previous-ssp token is not set to 1. If in 32-bit/compatibility mode and SSP recorded in previous-ssp token is beyond 4G.
#PF(fault-code)	If a page fault occurs.

Real-Address Mode Exceptions

#UD	The SAVEPREVSSP instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The SAVEPREVSSP instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

#UD	If the LOCK prefix is used. If CR4.CET = 0. If CPL = 3 and IA32_U_CET.SH_STK_EN = 0. If CPL < 3 and IA32_S_CET.SH_STK_EN = 0.
#GP(0)	If SSP not 8 byte aligned. If carry flag is set. If bit 1 of the previous-ssp token is not set to 1.
#PF(fault-code)	If a page fault occurs.

SETSSBSY—Mark Shadow Stack Busy

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 01 E8 SETSSBSY	Z0	V/V	CET_SS	Set busy flag in supervisor shadow stack token reference by IA32_PLO_SSP.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

The SETSSBSY instruction verifies the presence of a non-busy supervisor shadow stack token at the address in the IA32_PLO_SSP MSR and marks it busy. Following successful execution of the instruction, the SSP is set to the value of the IA32_PLO_SSP MSR.

This instruction cannot be executed when FRED transitions are enabled. FRED transitions do not use supervisor shadow stack tokens.

Operation

IF CR4.CET = 0 OR CR4.FRED = 1
THEN #UD; FI;

IF IA32_S_CET.SH_STK_EN = 0
THEN #UD; FI;

IF CPL > 0
THEN GP(0); FI;

SSP_LA = IA32_PLO_SSP

If SSP_LA not aligned to 8 bytes
THEN #GP(0); FI;

expected_token_value = SSP_LA (* busy bit must not be set *)

new_token_value = SSP_LA | BUSY_BIT (* set busy bit; bit position 0 *)

IF shadow_stack_lock_cmpxchg8B(SSP_LA, new_token_value, expected_token_value) != expected_token_value
THEN #CP(SETSSBSY); FI;

SSP = SSP_LA

Flags Affected

None.

C/C++ Compiler Intrinsic Equivalent

SETSSBSYvoid _setssbsy(void);

Protected Mode Exceptions

#UD If the LOCK prefix is used.

If CR4.CET = 0.

IF IA32_S_CET.SH_STK_EN = 0.

#GP(0) If IA32_PLO_SSP not aligned to 8 bytes.

If CPL is not 0.

#CP(setssbsy)	If busy bit in token is set. If the address in token is not below 4 GBytes.
#PF(fault-code)	If a page fault occurs.

Real-Address Mode Exceptions

#UD	The SETSSBSY instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The SETSSBSY instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	If the LOCK prefix is used. If CR4.CET = 0. If IA32_S_CET.SH_STK_EN = 0. If CR4.FRED = 1.
#GP(0)	If IA32_PL0_SSP not aligned to 8 bytes. If CPL is not 0.
#CP(setssbsy)	If busy bit in token is set. If the address in token is not below 4 GBytes.
#PF(fault-code)	If a page fault occurs.

64-Bit Mode Exceptions

Same as [compatibility](#) mode exceptions (except there is no check on the token address).

SWAPGS—Swap GS Base Register

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 01 F8	SWAPGS	Z0	Valid	Invalid	Exchanges the current GS base register value with the IA32_KERNEL_GS_BASE MSR .

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

SWAPGS exchanges the current GS base register value with [the IA32_KERNEL_GS_BASE MSR \(MSR address C0000102H\)](#). The SWAPGS instruction is a privileged instruction intended for use by system software.

When using SYSCALL to implement system calls, there is no kernel stack at the OS entry point. Neither is there a straightforward method to obtain a pointer to kernel structures from which the kernel stack pointer could be read. Thus, the kernel cannot save general purpose registers or reference memory.

By design, SWAPGS does not require any general purpose registers or memory operands. No registers need to be saved before using the instruction. SWAPGS exchanges the CPL 0 data pointer from the [IA32_KERNEL_GS_BASE MSR](#) with the GS base register. The kernel can then use the GS prefix on normal memory references to access kernel data structures. Similarly, when the OS kernel is entered using an interrupt or exception (where the kernel stack is already set up), SWAPGS can be used to quickly get a pointer to the kernel data structures.

The [IA32_KERNEL_GS_BASE MSR](#) itself is only accessible using RDMSR/WRMSR instructions. Those instructions are only accessible at privilege level 0. The WRMSR instruction ensures that the [IA32_KERNEL_GS_BASE MSR](#) contains a canonical address.

[The instruction cannot be executed when FRED transitions are enabled. FRED transitions perform the same swap when changing the CPL.](#)

Operation

```
IF CS.L ≠ 1 (* Not in 64-Bit Mode *) OR CR4.FRED = 1
    THEN
        #UD; FI;
IF CPL ≠ 0
    THEN #GP(0); FI;
tmp := GS.base;
GS.base := IA32_KERNEL_GS_BASE;
IA32_KERNEL_GS_BASE := tmp;
```

Flags Affected

None.

Protected Mode Exceptions

#UD [The SWAPGS instruction is not recognized in protected mode.](#)

Real-Address Mode Exceptions

#UD [The SWAPGS instruction is not recognized in real-address mode.](#)

Virtual-8086 Mode Exceptions

#UD [The SWAPGS instruction is not recognized in virtual-8086 mode.](#)

Compatibility Mode Exceptions

#UD The SWAPGS instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0) If CPL ≠ 0.
#UD If the LOCK prefix is used.
 If CR4.FRED = 1.

SYSCALL—Fast System Call

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 05	SYSCALL	Z0	Valid	Invalid	Fast call to privilege level 0 system procedures.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

SYSCALL invokes an OS system-call handler at privilege level 0. Its operation depends on whether FRED transitions are enabled.

Operation When FRED Transitions Are Not Enabled

When FRED transitions are not enabled, SYSCALL invokes the OS handler by loading RIP from the IA32_LSTAR MSR (after saving the address of the instruction following SYSCALL into RCX). (MSR writes ensure that the IA32_LSTAR MSR always contain a canonical address.) The OS handler returns using the SYSRET instruction.

SYSCALL also saves RFLAGS into R11 and then masks RFLAGS using the IA32_FMASK MSR (MSR address C0000084H); specifically, the processor clears in RFLAGS every bit corresponding to a bit that is set in the IA32_FMASK MSR.

SYSCALL loads the CS and SS selectors with values derived from bits 47:32 of the IA32_STAR MSR. However, the CS and SS descriptor caches are **not** loaded from the descriptors (in GDT or LDT) referenced by those selectors. Instead, the descriptor caches are loaded with fixed values. See the Operation section for details. It is the responsibility of OS software to ensure that the descriptors (in GDT or LDT) referenced by those selector values correspond to the fixed values loaded into the descriptor caches; the SYSCALL instruction does not ensure this correspondence.

The SYSCALL instruction does not save the stack pointer (RSP). If the OS system-call handler will change the stack pointer, it is the responsibility of software to save the previous value of the stack pointer. This might be done prior to executing SYSCALL, with software restoring the stack pointer with the instruction following SYSCALL (which will be executed after SYSRET). Alternatively, the OS system-call handler may save the stack pointer and restore it before executing SYSRET.

When shadow stacks are enabled at a privilege level where the SYSCALL instruction is invoked, the SSP is saved to the IA32_PL3_SSP MSR. If shadow stacks are enabled at privilege level 0, the SSP is loaded with 0. Refer to Chapter 6, "Procedure Calls, Interrupts, and Exceptions," and Chapter 18, "Control-flow Enforcement Technology (CET)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for additional CET details.

When FRED Transitions Are Enabled

When FRED transitions are enabled, SYSCALL invokes the OS handler by performing FRED event delivery. See Section 8.3, "FRED Event Delivery," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3. The event is delivered using event type 7 and vector 1. With FRED transitions, the OS handler uses the ERETU instruction to return to calling code operating at CPL 3.

Instruction ordering. Instructions following a SYSCALL may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the SYSCALL have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Operation

```
IF IA32_EFER.LMA = 0 OR CS.L = 0  (* SYSCALL can be used only in 64-bit mode *)  
  THEN #UD;  
ELSE IF CR4.FRED = 0
```

```

THEN
  IF IA32_EFER.SCE = 0
  THEN #UD;
  ELSE
    RCX := RIP;                                (* Will contain address of next instruction *)
    RIP := IA32_LSTAR;
    R11 := RFLAGS;
    RFLAGS := RFLAGS AND NOT(IA32_FMASK);

    CS.Selector := IA32_STAR[47:32] AND FFFCH  (* Operating system provides CS; RPL forced to 0 *)
    (* Set rest of CS to a fixed value *)
    CS.Base := 0;                                (* Flat segment *)
    CS.Limit := FFFFFFFH;                        (* With 4-KByte granularity, implies a 4-GByte limit *)
    CS.Type := 11;                              (* Execute/read code, accessed *)
    CS.S := 1;
    CS.DPL := 0;
    CS.P := 1;
    CS.L := 1;                                (* Entry is to 64-bit mode *)
    CS.D := 0;                                (* Required if CS.L = 1 *)
    CS.G := 1;                                (* 4-KByte granularity *)

    IF ShadowStackEnabled(CPL)
    THEN (* adjust so bits 63:N get the value of bit N-1, where N is the CPU's maximum linear-address width *)
      IA32_PL3_SSP := LA_adjust(SSP);
      (* With shadow stacks enabled the system call is supported from Ring 3 to Ring 0 *)
      (* OS supporting Ring 0 to Ring 0 system calls or Ring 1/2 to ring 0 system call *)
      (* Must preserve the contents of IA32_PL3_SSP to avoid losing ring 3 state *)
    FI;

    CPL := 0;

    IF ShadowStackEnabled(CPL)
    SSP := 0;
    FI;
    IF EndbranchEnabled(CPL)
    IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
    IA32_S_CET.SUPPRESS = 0
    FI;

    SS.Selector := IA32_STAR[47:32] + 8;        (* SS just above CS *)
    (* Set rest of SS to a fixed value *)
    SS.Base := 0;                                (* Flat segment *)
    SS.Limit := FFFFFFFH;                        (* With 4-KByte granularity, implies a 4-GByte limit *)
    SS.Type := 3;                              (* Read/write data, accessed *)
    SS.S := 1;
    SS.DPL := 0;
    SS.P := 1;
    SS.B := 1;                                (* 32-bit stack segment *)
    SS.G := 1;                                (* 4-KByte granularity *)

    FI;
  ELSE (* CR4.FRED = 1 *)
    FRED event delivery of SYSCALL;            (* save instruction length on stack *)
  FI;

```


Flags Affected

All.

Protected Mode Exceptions

#UD The SYSCALL instruction is not recognized in protected mode.

Real-Address Mode Exceptions

#UD The SYSCALL instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The SYSCALL instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD The SYSCALL instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

- #UD If IA32_EFER.SCE = 0.
 If the LOCK prefix is used.
- #GP(0) If FRED event delivery of SYSCALL would load RIP with a non-canonical address.
 If FRED event delivery of SYSCALL would load SSP with an address that is not 8-byte aligned.
 If a shadow-stack access performed by FRED event delivery of SYSCALL uses a non-canonical address or causes a LASS violation.
- #SS(0) If an ordinary stack access performed by FRED event delivery of SYSCALL uses a non-canonical address or causes a LASS violation.
- #PF If a memory access performed by FRED event delivery of SYSCALL causes a page fault.

SYSENTER—Fast System Call

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 34	SYSENTER	Z0	Valid	Valid	Fast call to privilege level 0 system procedures.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

SYSENTER invokes an OS system-call handler at privilege level 0. Its operation depends on whether FRED transitions are enabled.

Operation When FRED Transitions Are Not Enabled

When FRED transitions are not enabled, SYSENTER is a companion instruction to SYSEXIT. SYSENTER is optimized to provide the maximum performance for system calls from user code running at privilege level 3 to operating system or executive procedures running at privilege level 0.

When executed in IA-32e mode, the SYSENTER instruction transitions the logical processor to 64-bit mode; otherwise, the logical processor remains in protected mode.

Prior to executing the SYSENTER instruction, software must specify the privilege level 0 code segment and code entry point, and the privilege level 0 stack segment and stack pointer by writing values to the following MSRs:

- **IA32_SYSENTER_CS** (MSR address 174H) — The lower 16 bits of this MSR are the segment selector for the privilege level 0 code segment. This value is also used to determine the segment selector of the privilege level 0 stack segment (see the Operation section). This value cannot indicate a null selector.
- **IA32_SYSENTER_EIP** (MSR address 176H) — The value of this MSR is loaded into RIP (thus, this value references the first instruction of the selected operating procedure or routine). In protected mode, only bits 31:0 are loaded.
- **IA32_SYSENTER_ESP** (MSR address 175H) — The value of this MSR is loaded into RSP (thus, this value contains the stack pointer for the privilege level 0 stack). This value cannot represent a non-canonical address. In protected mode, only bits 31:0 are loaded.

MSR writes ensure that the IA32_SYSENTER_EIP and IA32_SYSENTER_ESP MSRs always contain canonical addresses.

While SYSENTER loads the CS and SS selectors with values derived from the IA32_SYSENTER_CS MSR, the CS and SS descriptor caches are **not** loaded from the descriptors (in GDT or LDT) referenced by those selectors. Instead, the descriptor caches are loaded with fixed values. See the Operation section for details. It is the responsibility of OS software to ensure that the descriptors (in GDT or LDT) referenced by those selector values correspond to the fixed values loaded into the descriptor caches; the SYSENTER instruction does not ensure this correspondence.

The SYSENTER instruction cannot be invoked from real-address mode.

The SYSENTER and SYSEXIT instructions are companion instructions, but they do not constitute a call/return pair. When executing a SYSENTER instruction, the processor does not save state information for the user code (e.g., the instruction pointer), and neither the SYSENTER nor the SYSEXIT instruction supports passing parameters on the stack.

To use the SYSENTER and SYSEXIT instructions as companion instructions for transitions between privilege level 3 code and privilege level 0 operating system procedures, the following conventions must be followed:

- The segment descriptors for the privilege level 0 code and stack segments and for the privilege level 3 code and stack segments must be contiguous in a descriptor table. This convention allows the processor to compute the segment selectors from the value entered in the SYSENTER_CS_MSR MSR.
- The fast system call “stub” routines executed by user code (typically in shared libraries or DLLs) must save the required return IP and processor state information if a return to the calling procedure is required. Likewise, the

operating system or executive procedures called with SYSENTER instructions must have access to and use this saved return and state information when returning to the user code.

The SYSENTER and SYSEXIT instructions were introduced into the IA-32 architecture in the Pentium II processor. The availability of these instructions on a processor is indicated with the SYSENTER/SYSEXIT present (SEP) feature flag returned to the EDX register by the CPUID instruction. An operating system that qualifies the SEP flag must also qualify the processor family and model to ensure that the SYSENTER/SYSEXIT instructions are actually present. For example:

```
IF CPUID SEP bit is set
  THEN IF (Family = 6) and (Model < 3) and (Stepping < 3)
    THEN
      SYSENTER/SYSEXIT_Not_Supported; FI;
    ELSE
      SYSENTER/SYSEXIT_Supported; FI;
  FI;
```

When the CPUID instruction is executed on the Pentium Pro processor (model 1), the processor returns a the SEP flag as set, but does not support the SYSENTER/SYSEXIT instructions.

When shadow stacks are enabled at privilege level where SYSENTER instruction is invoked, the SSP is saved to the IA32_PL3_SSP MSR. If shadow stacks are enabled at privilege level 0, the SSP is loaded with 0. Refer to Chapter 6, "Procedure Calls, Interrupts, and Exceptions," and Chapter 18, "Control-flow Enforcement Technology (CET)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for additional CET details.

When FRED Transitions are Enabled

When FRED transitions are enabled, SYSENTER invokes the OS handler by performing FRED event delivery. See Section 8.3, "FRED Event Delivery," in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3. The event is delivered using event type 7 and vector 2. With FRED transitions, the OS handler uses the ERETU instruction to return to calling code operating at CPL 3.

Instruction ordering. Instructions following a SYSENTER may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the SYSENTER have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Operation

```
IF CR0.PE = 0 OR (CR4.FRED = 0 AND IA32_SYSENTER_CS[15:2] = 0)
  THEN #GP(0); FI;
IF CR4.FRED = 0
  THEN
    RFLAGS.VM := 0; (* Ensures protected mode execution *)
    RFLAGS.IF := 0; (* Mask interrupts *)
    IF in IA-32e mode
      THEN
        RSP := IA32_SYSENTER_ESP;
        RIP := IA32_SYSENTER_EIP;
      ELSE
        ESP := IA32_SYSENTER_ESP[31:0];
        EIP := IA32_SYSENTER_EIP[31:0];
    FI;
    CS.Selector := IA32_SYSENTER_CS[15:0] AND FFFCH;
    (* Operating system provides CS; RPL forced to 0 *)
    (* Set rest of CS to a fixed value *)
    CS.Base := 0; (* Flat segment *)
    CS.Limit := FFFFFFFH; (* With 4-KByte granularity, implies a 4-GByte limit *)
    CS.Type := 11; (* Execute/read code, accessed *)
```

```

CS.S := 1;
CS.DPL := 0;
CS.P := 1;
IF in IA-32e mode
    THEN
        CS.L := 1;                (* Entry is to 64-bit mode *)
        CS.D := 0;                (* Required if CS.L = 1 *)
    ELSE
        CS.L := 0;
        CS.D := 1;                (* 32-bit code segment*)
FI;
CS.G := 1;                        (* 4-KByte granularity *)

IF ShadowStackEnabled(CPL)
    THEN
        IF IA32_EFER.LMA = 0
            THEN IA32_PL3_SSP := SSP;
            ELSE (* adjust so bits 63:N get the value of bit N-1, where N is the CPU's maximum linear-address width *)
                IA32_PL3_SSP := LA_adjust(SSP);
        FI;
FI;

CPL := 0;

IF ShadowStackEnabled(CPL)
    SSP := 0;
FI;
IF EndbranchEnabled(CPL)
    IA32_S_CET.TRACKER = WAIT_FOR_ENDBRANCH
    IA32_S_CET.SUPPRESS = 0
FI;
SS.Selector := CS.Selector + 8;    (* SS just above CS *)
(* Set rest of SS to a fixed value *)
SS.Base := 0;                    (* Flat segment *)
SS.Limit := FFFFFFFH;            (* With 4-KByte granularity, implies a 4-GByte limit *)
SS.Type := 3;                    (* Read/write data, accessed *)
SS.S := 1;
SS.DPL := 0;
SS.P := 1;
SS.B := 1;                       (* 32-bit stack segment*)
SS.G := 1;                       (* 4-KByte granularity *)
ELSE (* CR4.FRED = 1 *)
    FRED event delivery of SYSENTER; (* save instruction length on stack *)
FI;

```

Flags Affected

VM, IF (see Operation above).

Protected Mode Exceptions

#GP(0) If IA32_SYSENTER_CS[15:2] = 0.
 #UD If the LOCK prefix is used.

Real-Address Mode Exceptions

- #GP The SYSENTER instruction is not recognized in real-address mode.
- #UD If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

- #UD If the LOCK prefix is used.
- #GP(0) If CR4.FRED = 0 and IA32_SYSENTER_CS[15:2] = 0.
If FRED event delivery of SYSENTER would load RIP with a non-canonical address.
If FRED event delivery of SYSENTER would load SSP with an address that is not 8-byte aligned.
If a shadow-stack access performed by FRED event delivery of SYSENTER uses a non-canonical address or causes a LASS violation.
- #SS(0) If an ordinary stack access performed by FRED event delivery of SYSENTER uses a non-canonical address or causes a LASS violation.
- #PF If a memory access performed by FRED event delivery of SYSENTER causes a page fault.

64-Bit Mode Exceptions

Same exceptions as in [compatibility](#) mode.

SYSEXIT—Fast Return from Fast System Call

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 35	SYSEXIT	Z0	Valid	Valid	Fast return to privilege level 3 user code.
REX.W + OF 35	SYSEXIT	Z0	Valid	Valid	Fast return to 64-bit mode privilege level 3 user code.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

Executes a fast return to privilege level 3 user code. SYSEXIT is a companion instruction to the SYSENTER instruction. The instruction is optimized to provide the maximum performance for returns from system procedures executing at protection levels 0 to user procedures executing at protection level 3. It must be executed from code executing at privilege level 0.

With a 64-bit operand size, SYSEXIT remains in 64-bit mode; otherwise, it either enters compatibility mode (if the logical processor is in IA-32e mode) or remains in protected mode (if it is not).

Prior to executing SYSEXIT, software must specify the privilege level 3 code segment and code entry point, and the privilege level 3 stack segment and stack pointer by writing values into the following MSR and general-purpose registers:

- **IA32_SYSENTER_CS** (MSR address 174H) — Contains a 32-bit value that is used to determine the segment selectors for the privilege level 3 code and stack segments (see the Operation section)
- **RDX** — The canonical address in this register is loaded into RIP (thus, this value references the first instruction to be executed in the user code). If the return is not to 64-bit mode, only bits 31:0 are loaded.
- **ECX** — The canonical address in this register is loaded into RSP (thus, this value contains the stack pointer for the privilege level 3 stack). If the return is not to 64-bit mode, only bits 31:0 are loaded.

The IA32_SYSENTER_CS MSR can be read from and written to using RDMSR and WRMSR.

While SYSEXIT loads the CS and SS selectors with values derived from the IA32_SYSENTER_CS MSR, the CS and SS descriptor caches are **not** loaded from the descriptors (in GDT or LDT) referenced by those selectors. Instead, the descriptor caches are loaded with fixed values. See the Operation section for details. It is the responsibility of OS software to ensure that the descriptors (in GDT or LDT) referenced by those selector values correspond to the fixed values loaded into the descriptor caches; the SYSEXIT instruction does not ensure this correspondence.

The SYSEXIT instruction can be invoked from all operating modes except real-address mode and virtual-8086 mode.

The SYSENTER and SYSEXIT instructions were introduced into the IA-32 architecture in the Pentium II processor. The availability of these instructions on a processor is indicated with the SYSENTER/SYSEXIT present (SEP) feature flag returned to the EDX register by the CPUID instruction. An operating system that qualifies the SEP flag must also qualify the processor family and model to ensure that the SYSENTER/SYSEXIT instructions are actually present. For example:

```
IF CPUID SEP bit is set
    THEN IF (Family = 6) and (Model < 3) and (Stepping < 3)
        THEN
            SYSENTER/SYSEXIT_Not_Supported; FI;
        ELSE
            SYSENTER/SYSEXIT_Supported; FI;
    FI;
```

When the CPUID instruction is executed on the Pentium Pro processor (model 1), the processor returns a the SEP flag as set, but does not support the SYSENTER/SYSEXIT instructions.

When shadow stacks are enabled at privilege level 3 the instruction loads SSP with value from IA32_PL3_SSP MSR. Refer to Chapter 7, “Interrupt and Exception Handling,” and Chapter 18, “Control-flow Enforcement Technology (CET),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for additional CET details.

The instruction cannot be executed when FRED transitions are enabled. An operating system that has enabled FRED transitions should use ERETU instead.

Instruction ordering. Instructions following a SYSEXIT may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the SYSEXIT have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Operation

IF CR4.FRED = 1

THEN #UD; FI;

IF IA32_SYSENTER_CS[15:2] = 0 OR CR0.PE = 0 OR CPL ≠ 0 THEN #GP(0); FI;

IF operand size is 64-bit

THEN (* Return to 64-bit mode *)

RSP := RCX;

RIP := RDX;

ELSE (* Return to protected mode or compatibility mode *)

RSP := ECX;

RIP := EDX;

FI;

IF operand size is 64-bit (* Operating system provides CS; RPL forced to 3 *)

THEN CS.Selector := IA32_SYSENTER_CS[15:0] + 32;

ELSE CS.Selector := IA32_SYSENTER_CS[15:0] + 16;

FI;

CS.Selector := CS.Selector OR 3; (* RPL forced to 3 *)

(* Set rest of CS to a fixed value *)

CS.Base := 0; (* Flat segment *)

CS.Limit := FFFFFFFH; (* With 4-KByte granularity, implies a 4-GByte limit *)

CS.Type := 11; (* Execute/read code, accessed *)

CS.S := 1;

CS.DPL := 3;

CS.P := 1;

IF operand size is 64-bit

THEN (* return to 64-bit mode *)

CS.L := 1; (* 64-bit code segment *)

CS.D := 0; (* Required if CS.L = 1 *)

ELSE (* return to protected mode or compatibility mode *)

CS.L := 0;

CS.D := 1; (* 32-bit code segment *)

FI;

CS.G := 1; (* 4-KByte granularity *)

CPL := 3;

IF ShadowStackEnabled(CPL)

THEN SSP := IA32_PL3_SSP;

FI;

SS.Selector := CS.Selector + 8; (* SS just above CS *)

(* Set rest of SS to a fixed value *)

SS.Base := 0; (* Flat segment *)

SS.Limit := FFFFFFFH; (* With 4-KByte granularity, implies a 4-GByte limit *)

```
SS.Type := 3;                                (* Read/write data, accessed *)
SS.S := 1;
SS.DPL := 3;
SS.P := 1;
SS.B := 1;                                (* 32-bit stack segment*)
SS.G := 1;                                (* 4-KByte granularity *)
```

Flags Affected

None.

Protected Mode Exceptions

```
#GP(0)      If IA32_SYSENTER_CS[15:2] = 0.
              If CPL ≠ 0.
#UD          If the LOCK prefix is used.
```

Real-Address Mode Exceptions

```
#GP          The SYSEXIT instruction is not recognized in real-address mode.
#UD          If the LOCK prefix is used.
```

Virtual-8086 Mode Exceptions

```
#GP(0)      The SYSEXIT instruction is not recognized in virtual-8086 mode.
```

Compatibility Mode Exceptions

```
#GP(0)      If IA32_SYSENTER_CS[15:2] = 0.
              If CPL ≠ 0.
#UD          If the LOCK prefix is used.
              If CR4.FRED = 1.
```

64-Bit Mode Exceptions

```
#GP(0)      If IA32_SYSENTER_CS = 0.
              If CPL ≠ 0.
              If RCX or RDX contains a non-canonical address.
#UD          If the LOCK prefix is used.
              If CR4.FRED = 1.
```


SYSRET—Return From Fast System Call

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 07	SYSRET	Z0	Valid	Invalid	Return to compatibility mode from fast system call.
REX.W + OF 07	SYSRET	Z0	Valid	Invalid	Return to 64-bit mode from fast system call.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

SYSRET is a companion instruction to the SYSCALL instruction. It returns from an OS system-call handler to user code at privilege level 3. It does so by loading RIP from RCX and loading RFLAGS from R11.¹ With a 64-bit operand size, SYSRET remains in 64-bit mode; otherwise, it enters compatibility mode and only the low 32 bits of the registers are loaded.

SYSRET loads the CS and SS selectors with values derived from bits 63:48 of the IA32_STAR MSR. However, the CS and SS descriptor caches are **not** loaded from the descriptors (in GDT or LDT) referenced by those selectors. Instead, the descriptor caches are loaded with fixed values. See the Operation section for details. It is the responsibility of OS software to ensure that the descriptors (in GDT or LDT) referenced by those selector values correspond to the fixed values loaded into the descriptor caches; the SYSRET instruction does not ensure this correspondence.

The SYSRET instruction does not modify the stack pointer (ESP or RSP). For that reason, it is necessary for software to switch to the user stack. The OS may load the user stack pointer (if it was saved after SYSCALL) before executing SYSRET; alternatively, user code may load the stack pointer (if it was saved before SYSCALL) after receiving control from SYSRET.

If the OS loads the stack pointer before executing SYSRET, it must ensure that the handler of any interrupt or exception delivered between restoring the stack pointer and successful execution of SYSRET is not invoked with the user stack. It can do so using approaches such as the following:

- External interrupts. The OS can prevent an external interrupt from being delivered by clearing EFLAGS.IF before loading the user stack pointer.
- Nonmaskable interrupts (NMIs). The OS can ensure that the NMI handler is invoked with the correct stack by using the interrupt stack table (IST) mechanism for gate 2 (NMI) in the IDT (see Section 7.14.5, “Interrupt Stack Table,” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A).
- General-protection exceptions (#GP). The SYSRET instruction generates #GP(0) if the value of RCX is not canonical. The OS can address this possibility using one or more of the following approaches:
 - Confirming that the value of RCX is canonical before executing SYSRET.
 - Using paging to ensure that the SYSCALL instruction will never save a non-canonical value into RCX.
 - Using the IST mechanism for gate 13 (#GP) in the IDT.

When shadow stacks are enabled at privilege level 3 the instruction loads SSP with value from IA32_PL3_SSP MSR. Refer to Chapter 6, “Procedure Calls, Interrupts, and Exceptions,” and Chapter 18, “Control-flow Enforcement Technology (CET),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for additional CET details.

The instruction cannot be executed when FRED transitions are enabled. An operating system that has enabled FRED transitions should use ERETU instead.

1. Regardless of the value of R11, the RF and VM flags are always 0 in RFLAGS after execution of SYSRET. In addition, all reserved bits in RFLAGS retain the fixed values.

Instruction ordering. Instructions following a SYSRET may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the SYSRET have completed execution (the later instructions may execute before data stored by the earlier instructions have become globally visible).

Operation

```

IF (CS.L  $\neq$  1 ) or (IA32_EFER.LMA  $\neq$  1) or (IA32_EFER.SCE  $\neq$  1) or (CR4.FRED = 1)
(* Not in 64-Bit Mode or SYSCALL/SYSRET not enabled in IA32_EFER or FRED enabled *)
    THEN #UD; FI;
IF (CPL  $\neq$  0) THEN #GP(0); FI;
IF (operand size is 64-bit)
    THEN (* Return to 64-Bit Mode *)
        IF (RCX is not canonical) THEN #GP(0);
        RIP := RCX;
    ELSE (* Return to Compatibility Mode *)
        RIP := ECX;
FI;
RFLAGS := (R11 & 3C7FD7H) | 2;          (* Clear RF, VM, reserved bits; set bit 1 *)
IF (operand size is 64-bit)
    THEN CS.Selector := IA32_STAR[63:48]+16;
    ELSE CS.Selector := IA32_STAR[63:48];
FI;
CS.Selector := CS.Selector OR 3;          (* RPL forced to 3 *)
(* Set rest of CS to a fixed value *)
CS.Base := 0;                            (* Flat segment *)
CS.Limit := FFFFFFFH;                    (* With 4-KByte granularity, implies a 4-GByte limit *)
CS.Type := 11;                           (* Execute/read code, accessed *)
CS.S := 1;
CS.DPL := 3;
CS.P := 1;
IF (operand size is 64-bit)
    THEN (* Return to 64-Bit Mode *)
        CS.L := 1;                        (* 64-bit code segment *)
        CS.D := 0;                        (* Required if CS.L = 1 *)
    ELSE (* Return to Compatibility Mode *)
        CS.L := 0;                        (* Compatibility mode *)
        CS.D := 1;                        (* 32-bit code segment *)
FI;
CS.G := 1;                                (* 4-KByte granularity *)
CPL := 3;
IF ShadowStackEnabled(CPL)
    SSP := IA32_PL3_SSP;
FI;
SS.Selector := (IA32_STAR[63:48]+8) OR 3; (* RPL forced to 3 *)
(* Set rest of SS to a fixed value *)
SS.Base := 0;                            (* Flat segment *)
SS.Limit := FFFFFFFH;                    (* With 4-KByte granularity, implies a 4-GByte limit *)
SS.Type := 3;                            (* Read/write data, accessed *)
SS.S := 1;
SS.DPL := 3;
SS.P := 1;
SS.B := 1;                               (* 32-bit stack segment*)
SS.G := 1;                               (* 4-KByte granularity *)

```

Flags Affected

All.

Protected Mode Exceptions

#UD The SYSRET instruction is not recognized in protected mode.

Real-Address Mode Exceptions

#UD The SYSRET instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The SYSRET instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD The SYSRET instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#UD If IA32_EFER.SCE = 0.
 If the LOCK prefix is used.
 If CR4.FRED = 1.

#GP(0) If CPL ≠ 0.
 If the return is to 64-bit mode and RCX contains a non-canonical address.

8. Updates to Chapter 5, Volume 2C

Change bars and text show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C: Instruction Set Reference*, V.

Changes to this chapter:

- Corrected register references in the Opcode/Instruction Table Description column for the following instructions:
 - VCVTUDQ2PD for EVEX.128 and EVEX.256
 - VCVTUDQ2PS for EVEX.256
 - VCVTUQQ2PS for EVEX.128 and EVEX.512
 -
- In the Opcode/Instruction Table Description column of the VCVTUSI2SS instruction, corrected integer type of the value being converted to unsigned.

5.1 TERNARY BIT VECTOR LOGIC TABLE

VPTERNLOGD/VPTERNLOGQ instructions operate on dword/qword elements and take three bit vectors of the respective input data elements to form a set of 32/64 indices, where each 3-bit value provides an index into an 8-bit lookup table represented by the imm8 byte of the instruction. The 256 possible values of the imm8 byte is constructed as a 16x16 boolean logic table. The 16 rows of the table uses the lower 4 bits of imm8 as row index. The 16 columns are referenced by imm8[7:4]. The 16 columns of the table are present in two halves, with 8 columns shown in Table 5-1 for the column index value between 0:7, followed by Table 5-2 showing the 8 columns corresponding to column index 8:15. This section presents the two-halves of the 256-entry table using a short-hand notation representing simple or compound boolean logic expressions with three input bit source data.

The three input bit source data will be denoted with the capital letters: A, B, C; where A represents a bit from the first source operand (also the destination operand), B and C represent a bit from the 2nd and 3rd source operands.

Each map entry takes the form of a logic expression consisting of one or more component expressions. Each component expression consists of either a unary or binary boolean operator and associated operands. Each binary boolean operator is expressed in lowercase letters, and operands concatenated after the logic operator. The unary operator 'not' is expressed using '!'. Additionally, the conditional expression "A?B:C" expresses a result returning B if A is set, returning C otherwise.

A binary boolean operator is followed by two operands, e.g., andAB. For a compound binary expression that contain commutative components and comprising the same logic operator, the 2nd logic operator is omitted and three operands can be concatenated in sequence, e.g., andABC. When the 2nd operand of the first binary boolean expression comes from the result of another boolean expression, the 2nd boolean expression is concatenated after the uppercase operand of the first logic expression, e.g., norBnandAC. When the result is independent of an operand, that operand is omitted in the logic expression, e.g., zeros or norCB.

The 3-input expression "majorABC" returns 0 if two or more input bits are 0, returns 1 if two or more input bits are 1. The 3-input expression "minorABC" returns 1 if two or more input bits are 0, returns 0 if two or more input bits are 1.

The building-block bit logic functions used in Table 5-1 and Table 5-2 include:

- Constants: TRUE (1), FALSE (0);
- Unary function: Not (!);
- Binary functions: and, nand, or, nor, xor, xnor;
- Conditional function: Select (?);
- Tertiary functions: major, minor.

Table 5-1. Lower 8 columns of the 16x16 Map of VPTERNLOG Boolean Logic Operations

Imm	[7:4]							
[3:0]	0H	1H	2H	3H	4H	5H	6H	7H
00H	FALSE	andAnorBC	norBnandAC	andA!B	norCnandBA	andA!C	andAxorBC	andAnandBC
01H	norABC	norCB	norBxorAC	A?!B:norBC	norCxorBA	A?!C:norBC	A?xorBC:norBC	A?nandBC:norBC
02H	andCnorBA	norBxnorAC	andC!B	norBnorAC	C?norBA:andBA	C?norBA:A	C?!B:andBA	C?!B:A
03H	norBA	norBandAC	C?!B:norBA	!B	C?norBA:xnorBA	A?!C:!B	A?xorBC:!B	A?nandBC:!B
04H	andBnorAC	norCxnorBA	B?norAC:andAC	B?norAC:A	andB!C	norCnorBA	B?!C:andAC	B?!C:A
05H	norCA	norCandBA	B?norAC:xnorAC	A?!B:!C	B?!C:norAC	!C	A?xorBC:!C	A?nandBC:!C
06H	norAxnorBC	A?norBC:xorBC	B?norAC:C	xorBorAC	C?norBA:B	xorCorBA	xorCB	B?!C:orAC
07H	norAandBC	minorABC	C?!B:!A	nandBorAC	B?!C:!A	nandCorBA	A?xorBC:nandBC	nandCB
08H	norAnandBC	A?norBC:andBC	andCxorBA	A?!B:andBC	andBxorAC	A?!C:andBC	A?xorBC:andBC	xorAandBC
09H	norAxorBC	A?norBC:xnorBC	C?xorBA:norBA	A?!B:xnorBC	B?xorAC:norAC	A?!C:xnorBC	xnorABC	A?nandBC:xnorBC
0AH	andC!A	A?norBC:C	andCnandBA	A?!B:C	C?!A:andBA	xorCA	xorCandBA	A?nandBC:C
0BH	C?!A:norBA	C?!A:!B	C?nandBA:norBA	C?nandBA:!B	B?xorAC:!A	B?xorAC:nandAC	C?nandBA:xnorBA	nandBxnorAC
0CH	andB!A	A?norBC:B	B?!A:andAC	xorBA	andBnandAC	A?!C:B	xorBandAC	A?nandBC:B
0DH	B?!A:norAC	B?!A:!C	B?!A:xnorAC	C?xorBA:nandBA	B?nandAC:norAC	B?nandAC:!C	B?nandAC:xnorAC	nandCxnorBA
0EH	norAnorBC	xorAorBC	B?!A:C	A?!B:orBC	C?!A:B	A?!C:orBC	B?nandAC:C	A?nandBC:orBC
0FH	!A	nandAorBC	C?nandBA:!A	nandBA	B?nandAC:!A	nandCA	nandAxnorBC	nandABC

Table 5-2 shows the half of 256-entry map corresponding to column index values 8:15.

Table 5-2. Upper 8 columns of the 16x16 Map of VPTERNLOG Boolean Logic Operations

Imm	[7:4]							
[3:0]	08H	09H	0AH	0BH	0CH	0DH	0EH	0FH
00H	andABC	andAxnorBC	andCA	B?andAC:A	andBA	C?andBA:A	andAorBC	A
01H	A?andBC:norBC	B?andAC:!C	A?C:norBC	C?A:!B	A?B:norBC	B?A:!C	xnorAorBC	orAnorBC
02H	andCxnorBA	B?andAC:xorAC	B?andAC:C	B?andAC:orAC	C?xnorBA:andBA	B?A:xorAC	B?A:C	B?A:orAC
03H	A?andBC:!B	xnorBandAC	A?C:!B	nandBnandAC	xnorBA	B?A:nandAC	A?orBC:!B	orA!B
04H	andBxnorAC	C?andBA:xorBA	B?xnorAC:andAC	B?xnorAC:A	C?andBA:B	C?andBA:orBA	C?A:B	C?A:orBA
05H	A?andBC:!C	xnorCandBA	xnorCA	C?A:nandBA	A?B:!C	nandCnandBA	A?orBC:!C	orA!C
06H	A?andBC:xorBC	xorABC	A?C:xorBC	B?xnorAC:orAC	A?B:xorBC	C?xnorBA:orBA	A?orBC:xorBC	orAxorBC
07H	xnorAandBC	A?xnorBC:nandBC	A?C:nandBC	nandBxorAC	A?B:nandBC	nandCxorBA	A?orBCnandBC	orAnandBC
08H	andCB	A?xnorBC:andBC	andCorAB	B?C:A	andBorAC	C?B:A	majorABC	orAandBC
09H	B?C:norAC	xnorCB	xnorCorBA	C?orBA:!B	xnorBorAC	B?orAC:!C	A?orBC:xnorBC	orAxnorBC
0AH	A?andBC:C	A?xnorBC:C	C	B?C:orAC	A?B:C	B?orAC:xorAC	orCandBA	orCA
0BH	B?C:!A	B?C:nandAC	orCnorBA	orC!B	B?orAC:!A	B?orAC:nandAC	orCxnorBA	nandBnorAC
0CH	A?andBC:B	A?xnorBC:B	A?C:B	C?orBA:xorBA	B	C?B:orBA	orBandAC	orBA
0DH	C?B!A	C?B:nandBA	C?orBA:!A	C?orBA:nandBA	orBnorAC	orB!C	orBxnorAC	nandCnorBA
0EH	A?andBC:orBC	A?xnorBC:orBC	A?C:orBC	orCxorBA	A?B:orBC	orBxorAC	orCB	orABC
0FH	nandAnandBC	nandAxorBC	orCIA	orCnandBA	orB!A	orBnandAC	nandAnorBC	TRUE

Table 5-1 and Table 5-2 translate each of the possible value of the imm8 byte to a Boolean expression. These tables can also be used by software to translate Boolean expressions to numerical constants to form the imm8 value needed to construct the VPTERNLOG syntax. There is a unique set of three byte constants (F0H, CCH, AAH) that can be used for this purpose as input operands in conjunction with the Boolean expressions defined in those tables. The reverse mapping can be expressed as:

Result_imm8 = Table_Lookup_Entry(0F0H, 0CCH, 0AAH)

Table_Lookup_Entry is the Boolean expression defined in Table 5-1 and Table 5-2.

5.2 INSTRUCTIONS (V)

Chapter 5 continues an alphabetical discussion of Intel® 64 and IA-32 instructions (V). See also: Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume

2A; Chapter 5, “Instruction Set Reference, V,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B; and Chapter 5, “Instruction Set Reference, V,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

VCVTUDQ2PD—Convert Packed Unsigned Doubleword Integers to Packed Double Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32 Bit Mode Support	CPUID Feature Flag	Description
EVEX.128.F3.0F.W0 7A /r VCVTUDQ2PD xmm1 {k1}{z}, xmm2/m64/m32bcst	A	V/V	(AVX512VL AND AVX512F) OR AVX10.1	Convert two packed unsigned doubleword integers from xmm2 /m64/m32bcst to packed double precision floating-point values in xmm1 with writemask k1.
EVEX.256.F3.0F.W0 7A /r VCVTUDQ2PD ymm1 {k1}{z}, xmm2/m128/m32bcst	A	V/V	(AVX512VL AND AVX512F) OR AVX10.1	Convert four packed unsigned doubleword integers from xmm2 /m128/m32bcst to packed double precision floating-point values in ymm1 with writemask k1.
EVEX.512.F3.0F.W0 7A /r VCVTUDQ2PD zmm1 {k1}{z}, ymm2/m256/m32bcst	A	V/V	AVX512F OR AVX10.1	Convert eight packed unsigned doubleword integers from ymm2 /m256/m32bcst to eight packed double precision floating-point values in zmm1 with writemask k1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Half	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

Converts packed unsigned doubleword integers in the source operand (second operand) to packed double precision floating-point values in the destination operand (first operand).

The source operand is a YMM/XMM/XMM (low 64 bits) register, a 256/128/64-bit memory location or a 256/128/64-bit vector broadcasted from a 32-bit memory location. The destination operand is a ZMM/YMM/XMM register conditionally updated with writemask k1.

Attempt to encode this instruction with EVEX embedded rounding is ignored.

Note: EVEX.vvvv is reserved and must be 1111b, otherwise instructions will #UD.

Operation

VCVTUDQ2PD (EVEX Encoded Versions) When SRC Operand is a Register

(KL, VL) = (2, 128), (4, 256), (8, 512)

FOR j := 0 TO KL-1

 i := j * 64

 k := j * 32

 IF k1[j] OR *no writemask*

 THEN DEST[i+63:i] :=

 Convert_UIInteger_To_Double_Precision_Floating_Point(SRC[k+31:k])

 ELSE

 IF *merging-masking* ; merging-masking

 THEN *DEST[i+63:i] remains unchanged*

 ELSE ; zeroing-masking

 DEST[i+63:i] := 0

 FI

FI;

ENDFOR

DEST[MAXVL-1:VL] := 0

VCVTUDQ2PD (EVEX Encoded Versions) When SRC Operand is a Memory Source

(KL, VL) = (2, 128), (4, 256), (8, 512)

```
FOR j := 0 TO KL-1
  i := j * 64
  k := j * 32
  IF k1[j] OR *no writemask*
    THEN
      IF (EVEX.b = 1)
        THEN
          DEST[i+63:i] :=
            Convert_UInteger_To_Double_Precision_Floating_Point(SRC[31:0])
        ELSE
          DEST[i+63:i] :=
            Convert_UInteger_To_Double_Precision_Floating_Point(SRC[k+31:k])
        FI;
      ELSE
        IF *merging-masking* ; merging-masking
          THEN *DEST[i+63:i] remains unchanged*
        ELSE ; zeroing-masking
          DEST[i+63:i] := 0
        FI
      FI;
    FI;
  ENDFOR
DEST[MAXVL-1:VL] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

```
VCVTUDQ2PD __m512d __mm512_cvtepu32_pd( __m256i a);
VCVTUDQ2PD __m512d __mm512_mask_cvtepu32_pd( __m512d s, __mmask8 k, __m256i a);
VCVTUDQ2PD __m512d __mm512_maskz_cvtepu32_pd( __mmask8 k, __m256i a);
VCVTUDQ2PD __m256d __mm256_cvtepu32_pd( __m128i a);
VCVTUDQ2PD __m256d __mm256_mask_cvtepu32_pd( __m256d s, __mmask8 k, __m128i a);
VCVTUDQ2PD __m256d __mm256_maskz_cvtepu32_pd( __mmask8 k, __m128i a);
VCVTUDQ2PD __m128d __mm_cvtepu32_pd( __m128i a);
VCVTUDQ2PD __m128d __mm_mask_cvtepu32_pd( __m128d s, __mmask8 k, __m128i a);
VCVTUDQ2PD __m128d __mm_maskz_cvtepu32_pd( __mmask8 k, __m128i a);
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

EVEX-encoded instructions, see Table 2-53, “Type E5 Class Exception Conditions.”

Additionally:

#UD If EVEX.vvvv != 1111B.

VCVTUDQ2PS—Convert Packed Unsigned Doubleword Integers to Packed Single Precision Floating-Point Values

Opcode/ Instruction	Op / En	64/32 Bit Mode Support	CPUID Feature Flag	Description
EVEX.128.F2.0F.W0 7A /r VCVTUDQ2PS xmm1 {k1}{z}, xmm2/m128/m32bcst	A	V/V	(AVX512VL AND AVX512F) OR AVX10.1	Convert four packed unsigned doubleword integers from xmm2/m128/m32bcst to packed single precision floating-point values in xmm1 with writemask k1.
EVEX.256.F2.0F.W0 7A /r VCVTUDQ2PS ymm1 {k1}{z}, ymm2/m256/m32bcst	A	V/V	(AVX512VL AND AVX512F) OR AVX10.1	Convert eight packed unsigned doubleword integers from ymm2/m256/m32bcst to packed single precision floating-point values in ymm1 with writemask k1.
EVEX.512.F2.0F.W0 7A /r VCVTUDQ2PS zmm1 {k1}{z}, zmm2/m512/m32bcst {er}	A	V/V	AVX512F OR AVX10.1	Convert sixteen packed unsigned doubleword integers from zmm2/m512/m32bcst to sixteen packed single precision floating-point values in zmm1 with writemask k1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

Converts packed unsigned doubleword integers in the source operand (second operand) to single precision floating-point values in the destination operand (first operand).

The source operand is a ZMM/YMM/XMM register, a 512/256/128-bit memory location or a 512/256/128-bit vector broadcasted from a 32-bit memory location. The destination operand is a ZMM/YMM/XMM register conditionally updated with writemask k1.

Note: EVEX.vvvv is reserved and must be 1111b, otherwise instructions will #UD.

Operation

VCVTUDQ2PS (EVEX Encoded Version) When SRC Operand is a Register

(KL, VL) = (4, 128), (8, 256), (16, 512)

IF (VL = 512) AND (EVEX.b = 1)

THEN

SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(EVEX.RC);

ELSE

SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(MXCSR.RC);

FI;

FOR j := 0 TO KL-1

i := j * 32

IF k1[j] OR *no writemask*

THEN DEST[i+31:i] :=

Convert_UInteger_To_Single_Precision_Floating_Point(SRC[i+31:i])

ELSE

IF *merging-masking* ; merging-masking

THEN *DEST[i+31:i] remains unchanged*

ELSE ; zeroing-masking

DEST[i+31:i] := 0

FI

```

    FI;
ENDFOR
DEST[MAXVL-1:VL] := 0

```

VCVTUDQ2PS (EVEX Encoded Version) When SRC Operand is a Memory Source

(KL, VL) = (4, 128), (8, 256), (16, 512)

```

FOR j := 0 TO KL-1
    i := j * 32
    IF k1[j] OR *no writemask*
        THEN
            IF (EVEX.b = 1)
                THEN
                    DEST[i+31:i] :=
                    Convert_UIInteger_To_Single_Precision_Floating_Point(SRC[31:0])
                ELSE
                    DEST[i+31:i] :=
                    Convert_UIInteger_To_Single_Precision_Floating_Point(SRC[i+31:i])
                FI;
            ELSE
                IF *merging-masking* ; merging-masking
                    THEN *DEST[i+31:i] remains unchanged*
                ELSE ; zeroing-masking
                    DEST[i+31:i] := 0
                FI
            FI;
        FI;
    ENDFOR
    DEST[MAXVL-1:VL] := 0

```

Intel C/C++ Compiler Intrinsic Equivalent

```

VCVTUDQ2PS __m512 __mm512_cvtepu32_ps( __m512i a);
VCVTUDQ2PS __m512 __mm512_mask_cvtepu32_ps( __m512 s, __mmask16 k, __m512i a);
VCVTUDQ2PS __m512 __mm512_maskz_cvtepu32_ps( __mmask16 k, __m512i a);
VCVTUDQ2PS __m512 __mm512_cvt_roundedu32_ps( __m512i a, int r);
VCVTUDQ2PS __m512 __mm512_mask_cvt_roundedu32_ps( __m512 s, __mmask16 k, __m512i a, int r);
VCVTUDQ2PS __m512 __mm512_maskz_cvt_roundedu32_ps( __mmask16 k, __m512i a, int r);
VCVTUDQ2PS __m256 __mm256_cvtepu32_ps( __m256i a);
VCVTUDQ2PS __m256 __mm256_mask_cvtepu32_ps( __m256 s, __mmask8 k, __m256i a);
VCVTUDQ2PS __m256 __mm256_maskz_cvtepu32_ps( __mmask8 k, __m256i a);
VCVTUDQ2PS __m128 __mm_cvtepu32_ps( __m128i a);
VCVTUDQ2PS __m128 __mm_mask_cvtepu32_ps( __m128 s, __mmask8 k, __m128i a);
VCVTUDQ2PS __m128 __mm_maskz_cvtepu32_ps( __mmask8 k, __m128i a);

```

SIMD Floating-Point Exceptions

Precision.

Other Exceptions

EVEX-encoded instructions, see Table 2-48, “Type E2 Class Exception Conditions.”

Additionally:

#UD If EVEX.vvvv != 1111B.

VCVTUQQ2PS—Convert Packed Unsigned Quadword Integers to Packed Single Precision Floating-Point Values

Opcode/ Instruction	Op / En	64/32 Bit Mode Support	CPUID Feature Flag	Description
EVEX.128.F2.0F.W1 7A /r VCVTUQQ2PS xmm1 {k1}{z}, xmm2/m128/m64bcst	A	V/V	(AVX512VL AND AVX512DQ) OR AVX10.1	Convert two packed unsigned quadword integers from xmm2/m128/m64bcst to packed single precision floating-point values in xmm1 with writemask k1.
EVEX.256.F2.0F.W1 7A /r VCVTUQQ2PS xmm1 {k1}{z}, ymm2/m256/m64bcst	A	V/V	(AVX512VL AND AVX512DQ) OR AVX10.1	Convert four packed unsigned quadword integers from ymm2/m256/m64bcst to packed single precision floating-point values in xmm1 with writemask k1.
EVEX.512.F2.0F.W1 7A /r VCVTUQQ2PS ymm1 {k1}{z}, zmm2/m512/m64bcst {er}	A	V/V	AVX512DQ OR AVX10.1	Convert eight packed unsigned quadword integers from zmm2/m512/m64bcst to eight packed single precision floating-point values in ymm1 with writemask k1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

Converts packed unsigned quadword integers in the source operand (second operand) to single precision floating-point values in the destination operand (first operand).

EVEX encoded versions: The source operand is a ZMM/YMM/XMM register or a 512/256/128-bit memory location. The destination operand is a YMM/XMM/XMM (low 64 bits) register conditionally updated with writemask k1.

Note: EVEX.vvvv is reserved and must be 1111b, otherwise instructions will #UD.

Operation

VCVTUQQ2PS (EVEX Encoded Version) When SRC Operand is a Register

(KL, VL) = (2, 128), (4, 256), (8, 512)

IF (VL = 512) AND (EVEX.b = 1)

THEN

SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(EVEX.RC);

ELSE

SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(MXCSR.RC);

FI;

FOR j := 0 TO KL-1

i := j * 32

k := j * 64

IF k1[j] OR *no writemask*

THEN DEST[i+31:i] :=

Convert_UQuadInteger_To_Single_Precision_Floating_Point(SRC[k+63:k])

ELSE

IF *merging-masking* ; merging-masking

THEN *DEST[i+31:i] remains unchanged*

ELSE ; zeroing-masking

DEST[i+31:i] := 0

FI

FI;

ENDFOR

DEST[MAXVL-1:VL/2] := 0

VCVTUQQ2PS (EVEX Encoded Version) When SRC Operand is a Memory Source

(KL, VL) = (2, 128), (4, 256), (8, 512)

```
FOR j := 0 TO KL-1
  i := j * 32
  k := j * 64
  IF k1[j] OR *no writemask*
    THEN
      IF (EVEX.b = 1)
        THEN
          DEST[i+31:i] :=
            Convert_UQuadInteger_To_Single_Precision_Floating_Point(SRC[63:0])
        ELSE
          DEST[i+31:i] :=
            Convert_UQuadInteger_To_Single_Precision_Floating_Point(SRC[k+63:k])
        FI;
      ELSE
        IF *merging-masking* ; merging-masking
          THEN *DEST[i+31:i] remains unchanged*
        ELSE ; zeroing-masking
          DEST[i+31:i] := 0
        FI
      FI;
    ENDIF;
  DEST[MAXVL-1:VL/2] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

```
VCVTUQQ2PS __m256 __mm512_cvtepu64_ps( __m512i a);
VCVTUQQ2PS __m256 __mm512_mask_cvtepu64_ps( __m256 s, __mmask8 k, __m512i a);
VCVTUQQ2PS __m256 __mm512_maskz_cvtepu64_ps( __mmask8 k, __m512i a);
VCVTUQQ2PS __m256 __mm512_cvt_roundedu64_ps( __m512i a, int r);
VCVTUQQ2PS __m256 __mm512_mask_cvt_roundedu64_ps( __m256 s, __mmask8 k, __m512i a, int r);
VCVTUQQ2PS __m256 __mm512_maskz_cvt_roundedu64_ps( __mmask8 k, __m512i a, int r);
VCVTUQQ2PS __m128 __mm256_cvtepu64_ps( __m256i a);
VCVTUQQ2PS __m128 __mm256_mask_cvtepu64_ps( __m128 s, __mmask8 k, __m256i a);
VCVTUQQ2PS __m128 __mm256_maskz_cvtepu64_ps( __mmask8 k, __m256i a);
VCVTUQQ2PS __m128 __mm_cvtepu64_ps( __m128i a);
VCVTUQQ2PS __m128 __mm_mask_cvtepu64_ps( __m128 s, __mmask8 k, __m128i a);
VCVTUQQ2PS __m128 __mm_maskz_cvtepu64_ps( __mmask8 k, __m128i a);
```

SIMD Floating-Point Exceptions

Precision.

Other Exceptions

EVEX-encoded instructions, see Table 2-48, “Type E2 Class Exception Conditions.”

Additionally:

#UD If EVEX.vvvv != 1111B.

VCVTUSI2SS—Convert Unsigned Integer to Scalar Single Precision Floating-Point Value

Opcode/ Instruction	Op/ En	64/32 Bit Mode Support	CPUID Feature Flag	Description
EVEX.LLIG.F3.0F.W0 7B /r VCVTUSI2SS xmm1, xmm2, r/m32{er}	A	V/V	AVX512F OR AVX10.1	Convert one unsigned doubleword integer from r/m32 to one single precision floating-point value in xmm1.
EVEX.LLIG.F3.0F.W1 7B /r VCVTUSI2SS xmm1, xmm2, r/m64{er}	A	V/N.E. ¹	AVX512F OR AVX10.1	Convert one unsigned quadword integer from r/m64 to one single precision floating-point value in xmm1.

NOTES:

1. For this specific instruction, EVEX.W in non-64 bit is ignored; the instruction behaves as if the W0 version is used.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	N/A

Description

Converts an unsigned doubleword integer (or unsigned quadword integer if operand size is 64 bits) in the source operand (second operand) to a single precision floating-point value in the destination operand (first operand). The source operand can be a general-purpose register or a memory location. The destination operand is an XMM register. The result is stored in the low doubleword of the destination operand. When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or the embedded rounding control bits.

The second source operand can be a general-purpose register or a 32/64-bit memory location. The first source and destination operands are XMM registers. Bits (127:32) of the XMM register destination are copied from corresponding bits in the first source operand. Bits (MAXVL-1:128) of the destination register are zeroed.

EVEX.W1 version: promotes the instruction to use 64-bit input value in 64-bit mode.

Operation

VCVTUSI2SS (EVEX Encoded Version)

IF (SRC2 *is register*) AND (EVEX.b = 1)

THEN

SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(EVEX.RC);

ELSE

SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(MXCSR.RC);

FI;

IF 64-Bit Mode And OperandSize = 64

THEN

DEST[31:0] := Convert_UInteger_To_Single_Precision_Floating_Point(SRC[63:0]);

ELSE

DEST[31:0] := Convert_UInteger_To_Single_Precision_Floating_Point(SRC[31:0]);

FI;

DEST[127:32] := SRC1[127:32]

DEST[MAXVL-1:128] := 0

Intel C/C++ Compiler Intrinsic Equivalent

```
VCVTUSI2SS __m128 _mm_cvtsi32_ss( __m128 s, unsigned a);  
VCVTUSI2SS __m128 _mm_cvt_roundsi32_ss( __m128 s, unsigned a, int r);  
VCVTUSI2SS __m128 _mm_cvtsi64_ss( __m128 s, unsigned __int64 a);  
VCVTUSI2SS __m128 _mm_cvt_roundsi64_ss( __m128 s, unsigned __int64 a, int r);
```

SIMD Floating-Point Exceptions

Precision.

Other Exceptions

See Table 2-50, “Type E3NF Class Exception Conditions.”

9. Updates to Chapter 7, Volume 2D

Change bars and text show changes to Chapter 7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D: Instruction Set Reference, W-Z*.

Changes to this chapter:

- Content for Flexible Return and Event Delivery (FRED) was added in the GETSEC[ENTERACCS] leaf function.
 - Updated the Initialization Status for Register State CR4 in Table 7-4, "Register State Initialization After GETSEC[ENTERACCS]".
 - Added CR4.FRED in the section Operation in a Uni-Processor Platform.
- The pseudocode in the Operation section of the GETSEC[EXITAC] leaf function was updated for FRED.

7.1 OVERVIEW

This chapter describes the Safer Mode Extensions (SMX) for the Intel 64 and IA-32 architectures. Safer Mode Extensions (SMX) provide a programming interface for system software to establish a measured environment within the platform to support trust decisions by end users. The measured environment includes:

- Measured launch of a system executive, referred to as a Measured Launched Environment (MLE)¹. The system executive may be based on a Virtual Machine Monitor (VMM), a measured VMM is referred to as MVMM².
- Mechanisms to ensure the above measurement is protected and stored in a secure location in the platform.
- Protection mechanisms that allow the VMM to control attempts to modify the VMM.

The measurement and protection mechanisms used by a measured environment are supported by the capabilities of an Intel® Trusted Execution Technology (Intel® TXT) platform:

- The SMX are the processor's programming interface in an Intel TXT platform.
- The chipset in an Intel TXT platform provides enforcement of the protection mechanisms.
- Trusted Platform Module (TPM) 1.2 in the platform provides platform configuration registers (PCRs) to store software measurement values.

7.2 SMX FUNCTIONALITY

SMX functionality is provided in an Intel 64 processor through the GETSEC instruction via leaf functions. The GETSEC instruction supports multiple leaf functions. Leaf functions are selected by the value in EAX at the time GETSEC is executed. Each GETSEC leaf function is documented separately in the reference pages with a unique mnemonic (even though these mnemonics share the same opcode, 0F 37).

7.2.1 Detecting and Enabling SMX

Software can detect support for SMX operation using the CPUID instruction. If software executes CPUID with 1 in EAX, a value of 1 in bit 6 of ECX indicates support for SMX operation (GETSEC is available), see CPUID instruction for the layout of feature flags of reported by CPUID.01H:ECX.

System software enables SMX operation by setting CR4.SMXE[Bit 14] = 1 before attempting to execute GETSEC. Otherwise, execution of GETSEC results in the processor signaling an invalid opcode exception (#UD).

If the CPUID SMX feature flag is clear (CPUID.01H:ECX[6] = 0), attempting to set CR4.SMXE[Bit 14] results in a general protection exception.

The IA32_FEATURE_CONTROL MSR (at address 03AH) provides feature control bits that configure operation of VMX and SMX. These bits are documented in Table 7-1.

Table 7-1. Layout of IA32_FEATURE_CONTROL

Bit Position	Description
0	Lock bit (0 = unlocked, 1 = locked). When set to '1' further writes to this MSR are blocked.
1	Enable VMX in SMX operation.
2	Enable VMX outside SMX operation.

1. See the Intel® Trusted Execution Technology Measured Launched Environment Programming Guide.
2. An MVMM is sometimes referred to as a measured launched environment (MLE). See the Intel® Trusted Execution Technology Measured Launched Environment Programming Guide.

Table 7-1. Layout of IA32_FEATURE_CONTROL

7:3	Reserved
14:8	SENTER Local Function Enables: When set, each bit in the field represents an enable control for a corresponding SENTER function.
15	SENTER Global Enable: Must be set to '1' to enable operation of GETSEC[SENTER].
16	Reserved
17	SGX Launch Control Enable: Must be set to '1' to enable runtime re-configuration of SGX Launch Control via the IA32_SGXLEPUBKEYHASHn MSR.
18	SGX Global Enable: Must be set to '1' to enable Intel SGX leaf functions.
19	Reserved
20	LMCE On: When set, system software can program the MSRs associated with LMCE to configure delivery of some machine check exceptions to a single logical processor.
63:21	Reserved

- Bit 0 is a lock bit. If the lock bit is clear, an attempt to execute VMXON will cause a general-protection exception. Attempting to execute GETSEC[SENTER] when the lock bit is clear will also cause a general-protection exception. If the lock bit is set, WRMSR to the IA32_FEATURE_CONTROL MSR will cause a general-protection exception. Once the lock bit is set, the MSR cannot be modified until a power-on reset. System BIOS can use this bit to provide a setup option for BIOS to disable support for VMX, SMX or both VMX and SMX.
- Bit 1 enables VMX in SMX operation (between executing the SENTER and SEXIT leaves of GETSEC). If this bit is clear, an attempt to execute VMXON in SMX will cause a general-protection exception if executed in SMX operation. Attempts to set this bit on logical processors that do not support both VMX operation (Chapter 7, "Safer Mode Extensions Reference") and SMX operation cause general-protection exceptions.
- Bit 2 enables VMX outside SMX operation. If this bit is clear, an attempt to execute VMXON will cause a general-protection exception if executed outside SMX operation. Attempts to set this bit on logical processors that do not support VMX operation cause general-protection exceptions.
- Bits 8 through 14 specify enabled functionality of the SENTER leaf function. Each bit in the field represents an enable control for a corresponding SENTER function. Only enabled SENTER leaf functionality can be used when executing SENTER.
- Bits 15 specify global enable of all SENTER functionalities.

7.2.2 SMX Instruction Summary

System software must first query for available GETSEC leaf functions by executing GETSEC[CAPABILITIES]. The CAPABILITIES leaf function returns a bit map of available GETSEC leaves. An attempt to execute an unsupported leaf index results in an undefined opcode (#UD) exception.

7.2.2.1 GETSEC[CAPABILITIES]

The SMX functionality provides an architectural interface for newer processor generations to extend SMX capabilities. Specifically, the GETSEC instruction provides a capability leaf function for system software to discover the available GETSEC leaf functions that are supported in a processor. Table 7-2 lists the currently available GETSEC leaf functions.

Table 7-2. GETSEC Leaf Functions

Index (EAX)	Leaf function	Description
0	CAPABILITIES	Returns the available leaf functions of the GETSEC instruction.
1	Undefined	Reserved
2	ENTERACCS	Enter
3	EXITAC	Exit
4	SENDER	Launch an MLE.
5	SEXIT	Exit the MLE.
6	PARAMETERS	Return SMX related parameter information.
7	SMCTRL	SMX mode control.
8	WAKEUP	Wake up sleeping processors in safer mode.
9 - (4G-1)	Undefined	Reserved

7.2.2.2 GETSEC[ENTERACCS]

The GETSEC[ENTERACCS] leaf enables authenticated code execution mode. The ENTERACCS leaf function performs an authenticated code module load using the chipset public key as the signature verification. ENTERACCS requires the existence of an Intel® Trusted Execution Technology capable chipset since it unlocks the chipset private configuration register space after successful authentication of the loaded module. The physical base address and size of the authenticated code module are specified as input register values in EBX and ECX, respectively.

While in the authenticated code execution mode, certain processor state properties change. For this reason, the time in which the processor operates in authenticated code execution mode should be limited to minimize impact on external system events.

Upon entry into, the previous paging context is disabled (since the authenticated code module image is specified with physical addresses and can no longer rely upon external memory-based page-table structures).

Prior to executing the GETSEC[ENTERACCS] leaf, system software must ensure the logical processor issuing GETSEC[ENTERACCS] is the boot-strap processor (BSP), as indicated by IA32_APIC_BASE.BSP = 1. System software must ensure other logical processors are in a suitable idle state and not marked as BSP.

The GETSEC[ENTERACCS] leaf may be used by different agents to load different authenticated code modules to perform functions related to different aspects of a measured environment, for example system software and Intel® TXT enabled BIOS may use more than one authenticated code modules.

7.2.2.3 GETSEC[EXITAC]

GETSEC[EXITAC] takes the processor out of authenticated code execution mode. When this instruction leaf is executed, the contents of the authenticated code execution area are scrubbed and control is transferred to the non-authenticated context defined by a near pointer passed with the GETSEC[EXITAC] instruction.

The authenticated code execution area is no longer accessible after completion of GETSEC[EXITAC]. RBX (or EBX) holds the address of the near absolute indirect target to be taken.

7.2.2.4 GETSEC[SENDER]

The GETSEC[SENDER] leaf function is used by the initiating logical processor (ILP) to launch an MLE. GETSEC[SENDER] can be considered a superset of the ENTERACCS leaf, because it enters as part of the measured environment launch.

Measured environment startup consists of the following steps:

- the ILP rendezvous the responding logical processors (RLPs) in the platform into a controlled state (At the completion of this handshake, all the RLPs except for the ILP initiating the measured environment launch are placed in a newly defined SENTER sleep state).
- Load and authenticate the authenticated code module required by the measured environment, and enter authenticated code execution mode.
- Verify and lock certain system configuration parameters.
- Measure the dynamic root of trust and store into the PCRs in TPM.
- Transfer control to the MLE with interrupts disabled.

Prior to executing the GETSEC[SENDER] leaf, system software must ensure the platform's TPM is ready for access and the ILP is the boot-strap processor (BSP), as indicated by IA32_APIC_BASE.BSP. System software must ensure other logical processors (RLPs) are in a suitable idle state and not marked as BSP.

System software launching a measurement environment is responsible for providing a proper authenticate code module address when executing GETSEC[SENDER]. The AC module responsible for the launch of a measured environment and loaded by GETSEC[SENDER] is referred to as SINIT. See *Intel® Trusted Execution Technology Measured Launched Environment Programming Guide* for additional information on system software requirements prior to executing GETSEC[SENDER].

7.2.2.5 GETSEC[SEXIT]

System software exits the measured environment by executing the instruction GETSEC[SEXIT] on the ILP. This instruction rendezvous the responding logical processors in the platform for exiting from the measured environment. External events (if left masked) are unmasked and Intel® TXT-capable chipset's private configuration space is re-locked.

7.2.2.6 GETSEC[PARAMETERS]

The GETSEC[PARAMETERS] leaf function is used to report attributes, options, and limitations of SMX operation. Software uses this leaf to identify operating limits or additional options.

The information reported by GETSEC[PARAMETERS] may require executing the leaf multiple times using EBX as an index. If the GETSEC[PARAMETERS] instruction leaf or if a specific parameter field is not available, then SMX operation should be interpreted to use the default limits of respective GETSEC leaves or parameter fields defined in the GETSEC[PARAMETERS] leaf.

7.2.2.7 GETSEC[SMCTRL]

The GETSEC[SMCTRL] leaf function is used for providing additional control over specific conditions associated with the SMX architecture. An input register is supported for selecting the control operation to be performed. See the specific leaf description for details on the type of control provided.

7.2.2.8 GETSEC[WAKEUP]

Responding logical processors (RLPs) are placed in the SENTER sleep state after the initiating logical processor executes GETSEC[SENDER]. The ILP can wake up RLPs to join the measured environment by using GETSEC[WAKEUP]. When the RLPs in SENTER sleep state wake up, these logical processors begin execution at the entry point defined in a data structure held in system memory (pointed to by an chipset register LT.MLE.JOIN) in TXT configuration space.

7.2.3 Measured Environment and SMX

This section gives a simplified view of a representative life cycle of a measured environment that is launched by a system executive using SMX leaf functions. The *Intel® Trusted Execution Technology Measured Launched Environment Programming Guide* provides more detailed examples of using SMX and chipset resources (including chipset registers, Trusted Platform Module) to launch an MVMM.

The life cycle starts with the system executive (an OS, an OS loader, and so forth) loading the MLE and SINIT AC module into available system memory. The system executive must validate and prepare the platform for the measured launch. When the platform is properly configured, the system executive executes GETSEC[SENDER] on the initiating logical processor (ILP) to rendezvous the responding logical processors into an SENTER sleep state, the ILP then enters into using the SINIT AC module. In a multi-threaded or multi-processing environment, the system executive must ensure that other logical processors are already in an idle loop, or asleep (such as after executing HLT) before executing GETSEC[SENDER].

After the GETSEC[SENDER] rendezvous handshake is performed between all logical processors in the platform, the ILP loads the chipset authenticated code module (SINIT) and performs an authentication check. If the check passes, the processor hashes the SINIT AC module and stores the result into TPM PCR 17. It then switches execution context to the SINIT AC module. The SINIT AC module will perform a number of platform operations, including: verifying the system configuration, protecting the system memory used by the MLE from I/O devices capable of DMA, producing a hash of the MLE, storing the hash value in TPM PCR 18, and various other operations. When SINIT completes execution, it executes the GETSEC[EXITAC] instruction and transfers control the MLE at the designated entry point.

Upon receiving control from the SINIT AC module, the MLE must establish its protection and isolation controls before enabling DMA and interrupts and transferring control to other software modules. It must also wake up the RLPs from their SENTER sleep state using the GETSEC[WAKEUP] instruction and bring them into its protection and isolation environment.

While executing in a measured environment, the MVMM can access the Trusted Platform Module (TPM) in locality 2. The MVMM has complete access to all TPM commands and may use the TPM to report current measurement values or use the measurement values to protect information such that only when the platform configuration registers (PCRs) contain the same value is the information released from the TPM. This protection mechanism is known as sealing.

A measured environment shutdown is ultimately completed by executing GETSEC[SEXIT]. Prior to this step system software is responsible for scrubbing sensitive information left in the processor caches, system memory.

7.3 GETSEC LEAF FUNCTIONS

This section provides detailed descriptions of each leaf function of the GETSEC instruction. GETSEC is available only if CPUID.01H:ECX[6] = 1. This indicates the availability of SMX and the GETSEC instruction. Before GETSEC can be executed, SMX must be enabled by setting CR4.SMXE[Bit 14] = 1.

A GETSEC leaf can only be used if it is shown to be available as reported by the GETSEC[CAPABILITIES] function. Attempts to access a GETSEC leaf index not supported by the processor, or if CR4.SMXE is 0, results in the signaling of an undefined opcode exception.

All GETSEC leaf functions are available in protected mode, including the compatibility sub-mode of IA-32e mode and the 64-bit sub-mode of IA-32e mode. Unless otherwise noted, the behavior of all GETSEC functions and interactions related to the measured environment are independent of IA-32e mode. This also applies to the interpretation of register widths¹ passed as input parameters to GETSEC functions and to register results returned as output parameters.

1. This chapter uses the 64-bit notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because processors that support SMX also support Intel 64 Architecture. The MVMM can be launched in IA-32e mode or outside IA-32e mode. The 64-bit notation of processor registers also refer to its 32-bit forms if SMX is used in 32-bit environment. In some places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

The GETSEC functions ENTERACCS, SENTER, SEXIT, and WAKEUP require a Intel® TXT capable-chipset to be present in the platform. The GETSEC[CAPABILITIES] returned bit vector in position 0 indicates an Intel® TXT-capable chipset has been sampled present¹ by the processor.

The processor's operating mode also affects the execution of the following GETSEC leaf functions: SMCTRL, ENTERACCS, EXITAC, SENTER, SEXIT, and WAKEUP. These functions are only allowed in protected mode at CPL = 0. They are not allowed while in SMM in order to prevent potential intra-mode conflicts. Further execution qualifications exist to prevent potential architectural conflicts (for example: nesting of the measured environment or authenticated code execution mode). See the definitions of the GETSEC leaf functions for specific requirements.

For the purpose of performance monitor counting, the execution of GETSEC functions is counted as a single instruction with respect to retired instructions. The response by a responding logical processor (RLP) to messages associated with GETSEC[SENDER] or GETSEC[SEXIT] is transparent to the retired instruction count on the ILP.

1. Sampled present means that the processor sent a message to the chipset and the chipset responded that it (a) knows about the message and (b) is capable of executing SENTER. This means that the chipset CAN support Intel® TXT, and is configured and WILLING to support it.

GETSEC[CAPABILITIES]—Report the SMX Capabilities

Opcode	Instruction	Description
NP OF 37 (EAX = 0)	GETSEC[CAPABILITIES]	Report the SMX capabilities. The capabilities index is input in EBX with the result returned in EAX.

Description

The GETSEC[CAPABILITIES] function returns a bit vector of supported GETSEC leaf functions. The CAPABILITIES leaf of GETSEC is selected with EAX set to 0 at entry. EBX is used as the selector for returning the bit vector field in EAX. GETSEC[CAPABILITIES] may be executed at all privilege levels, but the CR4.SMXE bit must be set or an undefined opcode exception (#UD) is returned.

With EBX = 0 upon execution of GETSEC[CAPABILITIES], EAX returns the a bit vector representing status on the presence of a Intel® TXT-capable chipset and the first 30 available GETSEC leaf functions. The format of the returned bit vector is provided in Table 7-3.

If bit 0 is set to 1, then an Intel® TXT-capable chipset has been sampled present by the processor. If bits in the range of 1-30 are set, then the corresponding GETSEC leaf function is available. If the bit value at a given bit index is 0, then the GETSEC leaf function corresponding to that index is unsupported and attempted execution results in a #UD.

Bit 31 of EAX indicates if further leaf indexes are supported. If the Extended Leafs bit 31 is set, then additional leaf functions are accessed by repeating GETSEC[CAPABILITIES] with EBX incremented by one. When the most significant bit of EAX is not set, then additional GETSEC leaf functions are not supported; indexing EBX to a higher value results in EAX returning zero.

Table 7-3. GETSEC Capability Result Encoding (EBX = 0)

Field	Bit position	Description
Chipset Present	0	Intel® TXT-capable chipset is present.
Undefined	1	Reserved
ENTERACCS	2	GETSEC[ENTERACCS] is available.
EXITAC	3	GETSEC[EXITAC] is available.
SENDER	4	GETSEC[SENDER] is available.
SEXIT	5	GETSEC[SEXIT] is available.
PARAMETERS	6	GETSEC[PARAMETERS] is available.
SMCTRL	7	GETSEC[SMCTRL] is available.
WAKEUP	8	GETSEC[WAKEUP] is available.
Undefined	30:9	Reserved
Extended Leafs	31	Reserved for extended information reporting of GETSEC capabilities.

Operation

```

IF (CR4.SMXE=0)
    THEN #UD;
ELSIF (in VMX non-root operation)
    THEN VM Exit (reason="GETSEC instruction");
IF (EBX=0) THEN
    BitVector := 0;
    IF (TXT chipset present)
        BitVector[Chipset present] := 1;
    IF (ENTERACCS Available)
        THEN BitVector[ENTERACCS] := 1;
    IF (EXITAC Available)
        THEN BitVector[EXITAC] := 1;
    IF (SENDER Available)
        THEN BitVector[SENDER] := 1;
    IF (SEXIT Available)
        THEN BitVector[SEXIT] := 1;
    IF (PARAMETERS Available)
        THEN BitVector[PARAMETERS] := 1;
    IF (SMCTRL Available)
        THEN BitVector[SMCTRL] := 1;
    IF (WAKEUP Available)
        THEN BitVector[WAKEUP] := 1;
    EAX := BitVector;
ELSE
    EAX := 0;
END;;

```

Flags Affected

None.

Use of Prefixes

LOCK	Causes #UD.
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ).
Operand size	Causes #UD.
NP	66/F2/F3 prefixes are not allowed.
Segment overrides	Ignored.
Address size	Ignored.
REX	Ignored.

Protected Mode Exceptions

#UD If CR4.SMXE = 0.

Real-Address Mode Exceptions

#UD If CR4.SMXE = 0.

Virtual-8086 Mode Exceptions

#UD If CR4.SMXE = 0.

Compatibility Mode Exceptions

#UD If CR4.SMXE = 0.

64-Bit Mode Exceptions

#UD If CR4.SMXE = 0.

VM-exit Condition

Reason (GETSEC) If in VMX non-root operation.

GETSEC[ENTERACCS]—Execute Authenticated Chipset Code

Opcode	Instruction	Description
NP OF 37 (EAX = 2)	GETSEC[ENTERACCS]	Enter authenticated code execution mode. EBX holds the authenticated code module physical base address. ECX holds the authenticated code module size (bytes).

Description

The GETSEC[ENTERACCS] function loads, authenticates, and executes an authenticated code module using an Intel® TXT platform chipset's public key. The ENTERACCS leaf of GETSEC is selected with EAX set to 2 at entry.

There are certain restrictions enforced by the processor for the execution of the GETSEC[ENTERACCS] instruction:

- Execution is not allowed unless the processor is in protected mode or IA-32e mode with CPL = 0 and EFLAGS.VM = 0.
- Processor cache must be available and not disabled, that is, CR0.CD and CR0.NW bits must be 0.
- For processor packages containing more than one logical processor, CR0.CD is checked to ensure consistency between enabled logical processors.
- For enforcing consistency of operation with numeric exception reporting using Interrupt 16, CR0.NE must be set.
- An Intel TXT-capable chipset must be present as communicated to the processor by sampling of the power-on configuration capability field after reset.
- The processor can not already be in authenticated code execution mode as launched by a previous GETSEC[ENTERACCS] or GETSEC[SENDER] instruction without a subsequent exiting using GETSEC[EXITAC]).
- To avoid potential operability conflicts between modes, the processor is not allowed to execute this instruction if it currently is in SMM or VMX operation.
- To ensure consistent handling of SIPI messages, the processor executing the GETSEC[ENTERACCS] instruction must also be designated the BSP (boot-strap processor) as defined by IA32_APIC_BASE.BSP (Bit 8).

Failure to conform to the above conditions results in the processor signaling a general protection exception.

Prior to execution of the ENTERACCS leaf, other logical processors, i.e., RLPs, in the platform must be:

- Idle in a wait-for-SIPI state (as initiated by an INIT assertion or through reset for non-BSP designated processors), or
- In the SENTER sleep state as initiated by a GETSEC[SENDER] from the initiating logical processor (ILP).

If other logical processor(s) in the same package are not idle in one of these states, execution of ENTERACCS signals a general protection exception. The same requirement and action applies if the other logical processor(s) of the same package do not have CR0.CD = 0.

A successful execution of ENTERACCS results in the ILP entering an authenticated code execution mode. Prior to reaching this point, the processor performs several checks. These include:

- Establish and check the location and size of the specified authenticated code module to be executed by the processor.
- Inhibit the ILP's response to the external events: INIT, A20M, NMI, and SMI.
- Broadcast a message to enable protection of memory and I/O from other processor agents.
- Load the designated code module into an authenticated code execution area.
- Isolate the contents of the authenticated code execution area from further state modification by external agents.
- Authenticate the authenticated code module.
- Initialize the initiating logical processor state based on information contained in the authenticated code module header.
- Unlock the Intel® TXT-capable chipset private configuration space and TPM locality 3 space.

- Begin execution in the authenticated code module at the defined entry point.

The GETSEC[ENTERACCS] function requires two additional input parameters in the general purpose registers EBX and ECX. EBX holds the authenticated code (AC) module physical base address (the AC module must reside below 4 GBytes in physical address space) and ECX holds the AC module size (in bytes). The physical base address and size are used to retrieve the code module from system memory and load it into the internal authenticated code execution area. The base physical address is checked to verify it is on a modulo-4096 byte boundary. The size is verified to be a multiple of 64, that it does not exceed the internal authenticated code execution area capacity (as reported by GETSEC[CAPABILITIES]), and that the top address of the AC module does not exceed 32 bits. An error condition results in an abort of the authenticated code execution launch and the signaling of a general protection exception.

As an integrity check for proper processor hardware operation, execution of GETSEC[ENTERACCS] will also check the contents of all the machine check status registers (as reported by the MSRs IA32_MCI_STATUS) for any valid uncorrectable error condition. In addition, the global machine check status register IA32_MCG_STATUS MCIP bit must be cleared and the IERR processor package pin (or its equivalent) must not be asserted, indicating that no machine check exception processing is currently in progress. These checks are performed prior to initiating the load of the authenticated code module. Any outstanding valid uncorrectable machine check error condition present in these status registers at this point will result in the processor signaling a general protection violation.

The ILP masks the response to the assertion of the external signals INIT#, A20M, NMI#, and SMI#. This masking remains active until optionally unmasked by GETSEC[EXITAC] (this defined unmasking behavior assumes GETSEC[ENTERACCS] was not executed by a prior GETSEC[SENDER]). The purpose of this masking control is to prevent exposure to existing external event handlers that may not be under the control of the authenticated code module.

The ILP sets an internal flag to indicate it has entered authenticated code execution mode. The state of the A20M pin is likewise masked and forced internally to a de-asserted state so that any external assertion is not recognized during authenticated code execution mode.

To prevent other (logical) processors from interfering with the ILP operating in authenticated code execution mode, memory (excluding implicit write-back transactions) access and I/O originating from other processor agents are blocked. This protection starts when the ILP enters into authenticated code execution mode. Only memory and I/O transactions initiated from the ILP are allowed to proceed. Exiting authenticated code execution mode is done by executing GETSEC[EXITAC]. The protection of memory and I/O activities remains in effect until the ILP executes GETSEC[EXITAC].

Prior to launching the authenticated execution module using GETSEC[ENTERACCS] or GETSEC[SENDER], the processor's MTRRs (Memory Type Range Registers) must first be initialized to map out the authenticated RAM addresses as WB (writeback). Failure to do so may affect the ability for the processor to maintain isolation of the loaded authenticated code module. If the processor detected this requirement is not met, it will signal an Intel® TXT reset condition with an error code during the loading of the authenticated code module.

While physical addresses within the load module must be mapped as WB, the memory type for locations outside of the module boundaries must be mapped to one of the supported memory types as returned by GETSEC[PARAMETERS] (or UC as default).

To conform to the minimum granularity of MTRR MSRs for specifying the memory type, authenticated code RAM (ACRAM) is allocated to the processor in 4096 byte granular blocks. If an AC module size as specified in ECX is not a multiple of 4096 then the processor will allocate up to the next 4096 byte boundary for mapping as ACRAM with indeterminate data. This pad area will not be visible to the authenticated code module as external memory nor can it depend on the value of the data used to fill the pad area.

At the successful completion of GETSEC[ENTERACCS], the architectural state of the processor is partially initialized from contents held in the header of the authenticated code module. The processor GDTR, CS, and DS selectors are initialized from fields within the authenticated code module. Since the authenticated code module must be relocatable, all address references must be relative to the authenticated code module base address in EBX. The processor GDTR base value is initialized to the AC module header field GDTBasePtr + module base address held in EBX and the GDTR limit is set to the value in the GDTLimit field. The CS selector is initialized to the AC module header SegSel field, while the DS selector is initialized to CS + 8. The segment descriptor fields are implicitly initialized to BASE=0, LIMIT=FFFFFh, G=1, D=1, P=1, S=1, read/write access for DS, and execute/read access for CS. The processor begins the authenticated code module execution with the EIP set to the AC module header EntryPoint field + module base address (EBX). The AC module based fields used for initializing the processor state are checked for consistency and any failure results in a shutdown condition.

A summary of the register state initialization after successful completion of GETSEC[ENTERACCS] is given for the processor in Table 7-4. The paging is disabled upon entry into authenticated code execution mode. The authenticated code module is loaded and initially executed using physical addresses. It is up to the system software after execution of GETSEC[ENTERACCS] to establish a new (or restore its previous) paging environment with an appropriate mapping to meet new protection requirements. EBP is initialized to the authenticated code module base physical address for initial execution in the authenticated environment. As a result, the authenticated code can reference EBP for relative address based references, given that the authenticated code module must be position independent.

Table 7-4. Register State Initialization After GETSEC[ENTERACCS]

Register State	Initialization Status	Comment
CR0	PG←0, AM←0, WP←0: Others unchanged	Paging, Alignment Check, Write-protection are disabled.
CR4	MCE←0, CET←0, PCIDE←0, FRED←0: Others unchanged	Machine Check Exceptions, Control-flow Enforcement Technology, Process-context Identifiers, and FRED disabled.
EFLAGS	00000002H	
IA32_EFER	0H	IA-32e mode disabled.
EIP	AC.base + EntryPoint	AC.base is in EBX as input to GETSEC[ENTERACCS].
[E R]BX	Pre-ENTERACCS state: Next [E R]IP prior to GETSEC[ENTERACCS]	Carry forward 64-bit processor state across GETSEC[ENTERACCS].
ECX	Pre-ENTERACCS state: [31:16]=GDTR.limit; [15:0]=CS.sel	Carry forward processor state across GETSEC[ENTERACCS].
[E R]DX	Pre-ENTERACCS state: GDTR base	Carry forward 64-bit processor state across GETSEC[ENTERACCS].
EBP	AC.base	
CS	Sel=[SegSel], base=0, limit=FFFFFh, G=1, D=1, AR=9BH	
DS	Sel=[SegSel] +8, base=0, limit=FFFFFh, G=1, D=1, AR=93H	
GDTR	Base= AC.base (EBX) + [GDTBasePtr], Limit=[GDTLimit]	
DR7	00000400H	
IA32_DEBUGCTL	0H	
IA32_MISC_ENABLE	See Table 7-5 for example.	The number of initialized fields may change due to processor implementation.
Performance counters and counter control registers	0H	

The segmentation related processor state that has not been initialized by GETSEC[ENTERACCS] requires appropriate initialization before use. Since a new GDT context has been established, the previous state of the segment selector values held in ES, SS, FS, GS, TR, and LDTR might not be valid.

The MSR IA32_EFER is also unconditionally cleared as part of the processor state initialized by ENTERACCS. Since paging is disabled upon entering authenticated code execution mode, a new paging environment will have to be reestablished in order to establish IA-32e mode while operating in authenticated code execution mode.

Debug exception and trap related signaling is also disabled as part of GETSEC[ENTERACCS]. This is achieved by resetting DR7, TF in EFLAGS, and the MSR IA32_DEBUGCTL. These debug functions are free to be re-enabled once supporting exception handler(s), descriptor tables, and debug registers have been properly initialized following entry into authenticated code execution mode. Also, any pending single-step trap condition will have been cleared upon entry into this mode.

Performance related counters and counter control registers are cleared as part of execution of ENTERACCS. This implies any active performance counters at any time of ENTERACCS execution will be disabled. To reactive the processor performance counters, this state must be re-initialized and re-enabled.

The IA32_MISC_ENABLE MSR is initialized upon entry into authenticated execution mode. Certain bits of this MSR are preserved because preserving these bits may be important to maintain previously established platform settings (See the footnote for Table 7-5.). The remaining bits are cleared for the purpose of establishing a more consistent environment for the execution of authenticated code modules. One of the impacts of initializing this MSR is any previous condition established by the MONITOR instruction will be cleared.

To support the possible return to the processor architectural state prior to execution of GETSEC[ENTERACCS], certain critical processor state is captured and stored in the general- purpose registers at instruction completion. [E|R]BX holds effective address ([E|R]IP) of the instruction that would execute next after GETSEC[ENTERACCS], ECX[15:0] holds the CS selector value, ECX[31:16] holds the GDTR limit field, and [E|R]DX holds the GDTR base field. The subsequent authenticated code can preserve the contents of these registers so that this state can be manually restored if needed, prior to exiting authenticated code execution mode with GETSEC[EXITAC]. For the processor state after exiting authenticated code execution mode, see the description of GETSEC[SEXIT].

Table 7-5. IA32_MISC_ENABLE MSR Initialization¹ by ENTERACCS and SENTER

Field	Bit position	Description
Fast strings enable	0	Clear to 0.
FOPCODE compatibility mode enable	2	Clear to 0.
Thermal monitor enable	3	Set to 1 if other thermal monitor capability is not enabled. ²
Split-lock disable	4	Clear to 0.
Bus lock on cache line splits disable	8	Clear to 0.
Hardware prefetch disable	9	Clear to 0.
GV1/2 legacy enable	15	Clear to 0.
MONITOR/MWAIT s/m enable	18	Clear to 0.
Adjacent sector prefetch disable	19	Clear to 0.

NOTES:

1. The number of IA32_MISC_ENABLE fields that are initialized may vary due to processor implementations.
2. ENTERACCS (and SENTER) initialize the state of processor thermal throttling such that at least a minimum level is enabled. If thermal throttling is already enabled when executing one of these GETSEC leaves, then no change in the thermal throttling control settings will occur. If thermal throttling is disabled, then it will be enabled via setting of the thermal throttle control bit 3 as a result of executing these GETSEC leaves.

The IDTR will also require reloading with a new IDT context after entering authenticated code execution mode, before any exceptions or the external interrupts INTR and NMI can be handled. Since external interrupts are re-enabled at the completion of authenticated code execution mode (as terminated with EXITAC), it is recommended that a new IDT context be established before this point. Until such a new IDT context is established, the programmer must take care in not executing an INT n instruction or any other operation that would result in an exception or trap signaling.

Prior to completion of the GETSEC[ENTERACCS] instruction and after successful authentication of the AC module, the private configuration space of the Intel TXT chipset is unlocked. The authenticated code module alone can gain access to this normally restricted chipset state for the purpose of securing the platform.

Once the authenticated code module is launched at the completion of GETSEC[ENTERACCS], it is free to enable interrupts by setting EFLAGS.IF and enable NMI by execution of IRET. This presumes that it has re-established interrupt handling support through initialization of the IDT, GDT, and corresponding interrupt handling code.

Operation in a Uni-Processor Platform

```
(* The state of the internal flag ACMODEFLAG persists across instruction boundary *)
IF (CR4.SMXE=0)
    THEN #UD;
ELSIF (in VMX non-root operation)
    THEN VM Exit (reason="GETSEC instruction");
ELSIF (GETSEC leaf unsupported)
    THEN #UD;
ELSIF ((in VMX operation) or
    (CR0.PE=0) or (CR0.CD=1) or (CR0.NW=1) or (CR0.NE=0) or
    (CPL>0) or (EFLAGS.VM=1) or
    (IA32_APIC_BASE.BSP=0) or
    (TXT chipset not present) or
    (ACMODEFLAG=1) or (IN_SMM=1))
    THEN #GP(0);
IF (GETSEC[PARAMETERS].Parameter_Type = 5, MCA_Handling (bit 6) = 0)
    FOR I = 0 to IA32_MCG_CAP.COUNT-1 DO
        IF (IA32_MC[I]_STATUS = uncorrectable error)
            THEN #GP(0);
    OD;
FI;
IF (IA32_MCG_STATUS.MCIP=1) or (IERR pin is asserted)
    THEN #GP(0);
ACBASE := EBX;
ACSIZE := ECX;
IF (((ACBASE MOD 4096) ≠ 0) or ((ACSIZE MOD 64) ≠ 0) or (ACSIZE < minimum module size) OR (ACSIZE > authenticated RAM
capacity)) or ((ACBASE+ACSIZE) > (232-1)))
    THEN #GP(0);
IF (secondary thread(s) CR0.CD = 1) or ((secondary thread(s) NOT(wait-for-SIPI)) and
    (secondary thread(s) not in SENTER sleep state)
    THEN #GP(0);
Mask SMI, INIT, A20M, and NMI external pin events;
IA32_MISC_ENABLE := (IA32_MISC_ENABLE & MASK_CONST*)
(* The hexadecimal value of MASK_CONST may vary due to processor implementations *)
A20M := 0;
IA32_DEBUGCTL := 0;
Invalidate processor TLB(s);
Drain Outgoing Transactions;
ACMODEFLAG := 1;
SignalTXTMessage(ProcessorHold);
Load the internal ACRAM based on the AC module size;
(* Ensure that all ACRAM loads hit Write Back memory space *)
IF (ACRAM memory type ≠ WB)
    THEN TXT-SHUTDOWN(#BadACMMType);
IF (AC module header version isnot supported) OR (ACRAM[ModuleType] ≠ 2)
    THEN TXT-SHUTDOWN(#UnsupportedACM);
(* Authenticate the AC Module and shutdown with an error if it fails *)
KEY := GETKEY(ACRAM, ACPBASE);
KEYHASH := HASH(KEY);
CSKEYHASH := READ(TXT.PUBLIC.KEY);
IF (KEYHASH ≠ CSKEYHASH)
    THEN TXT-SHUTDOWN(#AuthenticateFail);
SIGNATURE := DECRYPT(ACRAM, ACPBASE, KEY);
(* The value of SIGNATURE_LEN_CONST is implementation-specific*)
```



```

FOR I=0 to SIGNATURE_LEN_CONST - 1 DO
    ACRAM[SCRATCH.I] := SIGNATURE[I];
COMPUTEDSIGNATURE := HASH(ACRAM, ACBASE, ACSIZE);
FOR I=0 to SIGNATURE_LEN_CONST - 1 DO
    ACRAM[SCRATCH.SIGNATURE_LEN_CONST+I] := COMPUTEDSIGNATURE[I];
IF (SIGNATURE ≠ COMPUTEDSIGNATURE)
    THEN TXT-SHUTDOWN(#AuthenticateFail);
ACMCONTROL := ACRAM[CodeControl];
IF ((ACMCONTROL.0 = 0) and (ACMCONTROL.1 = 1) and (snoop hit to modified line detected on ACRAM load))
    THEN TXT-SHUTDOWN(#UnexpectedHITM);
IF (ACMCONTROL reserved bits are set)
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACRAM[GDTBasePtr] < (ACRAM[HeaderLen] * 4 + Scratch_size)) OR
    ((ACRAM[GDTBasePtr] + ACRAM[GDTLimit]) >= ACSIZE))
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACMCONTROL.0 = 1) and (ACMCONTROL.1 = 1) and (snoop hit to modified line detected on ACRAM load))
    THEN ACEntryPoint := ACBASE+ACRAM[ErrorEntryPoint];
ELSE
    ACEntryPoint := ACBASE+ACRAM[EntryPoint];
IF ((ACEntryPoint >= ACSIZE) OR (ACEntryPoint < (ACRAM[HeaderLen] * 4 + Scratch_size))) THEN TXT-SHUTDOWN(#BadACMFormat);
IF (ACRAM[GDTLimit] & FFFF0000h)
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACRAM[SegSel] > (ACRAM[GDTLimit] - 15)) OR (ACRAM[SegSel] < 8))
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACRAM[SegSel].TI=1) OR (ACRAM[SegSel].RPL≠0))
    THEN TXT-SHUTDOWN(#BadACMFormat);
CR0.[PG.AM.WP] := 0;
CR4.MCE := 0;
ACRAM[CR4High].FRED := CR4.FRED;
CR4.FRED := 0;
EFLAGS := 00000002h;
IA32_EFER := 0h;
[E|R]BX := [E|R]IP of the instruction after GETSEC[ENTERACCS];
ECX := Pre-GETSEC[ENTERACCS] GDT.limit:CS.sel;
[E|R]DX := Pre-GETSEC[ENTERACCS] GDT.base;
EBP := ACBASE;
GDTR.BASE := ACBASE+ACRAM[GDTBasePtr];
GDTR.LIMIT := ACRAM[GDTLimit];
CS.SEL := ACRAM[SegSel];
CS.BASE := 0;
CS.LIMIT := FFFFFFFh;
CS.G := 1;
CS.D := 1;
CS.AR := 9Bh;
DS.SEL := ACRAM[SegSel]+8;
DS.BASE := 0;
DS.LIMIT := FFFFFFFh;
DS.G := 1;
DS.D := 1;
DS.AR := 93h;
DR7 := 00000400h;
IA32_DEBUGCTL := 0;
SignalTXTMsg(OpenPrivate);
SignalTXTMsg(OpenLocality3);

```

```
EIP := ACEntryPoint;
END;
```

Flags Affected

All flags are cleared.

Use of Prefixes

LOCK	Causes #UD.
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ).
Operand size	Causes #UD.
NP	66/F2/F3 prefixes are not allowed.
Segment overrides	Ignored.
Address size	Ignored.
REX	Ignored.

Protected Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[ENTERACCS] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.CD = 1 or CR0.NW = 1 or CR0.NE = 0 or CR0.PE = 0 or CPL > 0 or EFLAGS.VM = 1. If a Intel® TXT-capable chipset is not present. If in VMX root operation. If the initiating processor is not designated as the bootstrap processor via the MSR bit IA32_APIC_BASE.BSP. If the processor is already in authenticated code execution mode. If the processor is in SMM. If a valid uncorrectable machine check error is logged in IA32_MC[I]_STATUS. If the authenticated code base is not on a 4096 byte boundary. If the authenticated code size > processor internal authenticated code area capacity. If the authenticated code size is not modulo 64. If other enabled logical processor(s) of the same package CR0.CD = 1. If other enabled logical processor(s) of the same package are not in the wait-for-SIPI or SENTER sleep state.

Real-Address Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[ENTERACCS] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[ENTERACCS] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[ENTERACCS] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[ENTERACCS] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

#GP	If AC code module does not reside in physical address below $2^{32} - 1$.

64-Bit Mode Exceptions

All protected mode exceptions apply.

#GP	If AC code module does not reside in physical address below $2^{32} - 1$.

VM-exit Condition

Reason (GETSEC)	If in VMX non-root operation.

GETSEC[EXITAC]—Exit Authenticated Code Execution Mode

Opcode	Instruction	Description
NP OF 37 (EAX=3)	GETSEC[EXITAC]	Exit authenticated code execution mode. RBX holds the Near Absolute Indirect jump target and EDX hold the exit parameter flags.

Description

The GETSEC[EXITAC] leaf function exits the ILP out of authenticated code execution mode established by GETSEC[ENTERACCS] or GETSEC[SENDER]. The EXITAC leaf of GETSEC is selected with EAX set to 3 at entry. EBX (or RBX, if in 64-bit mode) holds the near jump target offset for where the processor execution resumes upon exiting authenticated code execution mode. EDX contains additional parameter control information. Currently only an input value of 0 in EDX is supported. All other EDX settings are considered reserved and result in a general protection violation.

GETSEC[EXITAC] can only be executed if the processor is in protected mode with CPL = 0 and EFLAGS.VM = 0. The processor must also be in authenticated code execution mode. To avoid potential operability conflicts between modes, the processor is not allowed to execute this instruction if it is in SMM or in VMX operation. A violation of these conditions results in a general protection violation.

Upon completion of the GETSEC[EXITAC] operation, the processor unmask responses to external event signals INIT#, NMI#, and SMI#. This unmasking is performed conditionally, based on whether the authenticated code execution mode was entered via execution of GETSEC[SENDER] or GETSEC[ENTERACCS]. If the processor is in authenticated code execution mode due to the execution of GETSEC[SENDER], then these external event signals will remain masked. In this case, A20M is kept disabled in the measured environment until the measured environment executes GETSEC[SEXIT]. INIT# is unconditionally unmasked by EXITAC. Note that any events that are pending, but have been blocked while in authenticated code execution mode, will be recognized at the completion of the GETSEC[EXITAC] instruction if the pin event is unmasked.

The intent of providing the ability to optionally leave the pin events SMI#, and NMI# masked is to support the completion of a measured environment bring-up that makes use of VMX. In this envisioned security usage scenario, these events will remain masked until an appropriate virtual machine has been established in order to field servicing of these events in a safer manner. Details on when and how events are masked and unmasked in VMX operation are described in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C. It should be cautioned that if no VMX environment is to be activated following GETSEC[EXITAC], that these events will remain masked until the measured environment is exited with GETSEC[SEXIT]. If this is not desired then the GETSEC function SMCTRL(0) can be used for unmasking SMI# in this context. NMI# can be correspondingly unmasked by execution of IRET.

A successful exit of the authenticated code execution mode requires the ILP to perform additional steps as outlined below:

- Invalidate the contents of the internal authenticated code execution area.
- Invalidate processor TLBs.
- Clear the internal processor AC Mode indicator flag.
- Re-lock the TPM locality 3 space.
- Unlock the Intel® TXT-capable chipset memory and I/O protections to allow memory and I/O activity by other processor agents.
- Perform a near absolute indirect jump to the designated instruction location.

The content of the authenticated code execution area is invalidated by hardware in order to protect it from further use or visibility. This internal processor storage area can no longer be used or relied upon after GETSEC[EXITAC]. Data structures need to be re-established outside of the authenticated code execution area if they are to be referenced after EXITAC. Since addressed memory content formerly mapped to the authenticated code execution area may no longer be coherent with external system memory after EXITAC, processor TLBs in support of linear to physical address translation are also invalidated.

Upon completion of GETSEC[EXITAC] a near absolute indirect transfer is performed with EIP loaded with the contents of EBX (based on the current operating mode size). In 64-bit mode, all 64 bits of RBX are loaded into RIP if REX.W precedes GETSEC[EXITAC]. Otherwise RBX is treated as 32 bits even while in 64-bit mode. Conventional CS limit checking is performed as part of this control transfer. Any exception conditions generated as part of this control transfer will be directed to the existing IDT; thus it is recommended that an IDTR should also be established prior to execution of the EXITAC function if there is a need for fault handling. In addition, any segmentation related (and paging) data structures to be used after EXITAC should be re-established or validated by the authenticated code prior to EXITAC.

In addition, any segmentation related (and paging) data structures to be used after EXITAC need to be re-established and mapped outside of the authenticated RAM designated area by the authenticated code prior to EXITAC. Any data structure held within the authenticated RAM allocated area will no longer be accessible after completion by EXITAC.

Operation

(* The state of the internal flag ACMODEFLAG and SENTERFLAG persist across instruction boundary *)

```

IF (CR4.SMXE=0)
    THEN #UD;
ELSIF ( in VMX non-root operation)
    THEN VM Exit (reason="GETSEC instruction");
ELSIF (GETSEC leaf unsupported)
    THEN #UD;
ELSIF ((in VMX operation) or ( in 64-bit mode) and ( RBX is non-canonical) )
    (CR0.PE=0) or (CPL>0) or (EFLAGS.VM=1) or
    (ACMODEFLAG=0) or (IN_SMM=1)) or (EDX ≠ 0))
    THEN #GP(0);
IF (OperandSize = 32)
    THEN tempEIP := EBX;
ELSIF (OperandSize = 64)
    THEN tempEIP := RBX;
ELSE
    tempEIP := EBX AND 0000FFFFH;
IF (tempEIP > code segment limit)
    THEN #GP(0);
IF (ACRAM[CR4High].FRED = 1) and (IA32_EFER.LMA = 0)
    THEN #GP(0);
ELSE CR4.FRED = ACRAM[CR4High].FRED;
Invalidate ACRAM contents;
Invalidate processor TLB(s);
Drain outgoing messages;
SignalTXTMsg(CloseLocality3);
SignalTXTMsg(LockSMRAM);
SignalTXTMsg(ProcessorRelease);
Unmask INIT;
IF (SENTERFLAG=0)
    THEN Unmask SMI, INIT, NMI, and A20M pin event;
ELSEIF (IA32_SMM_MONITOR_CTL[0] = 0)
    THEN Unmask SMI pin event;
ACMODEFLAG := 0;
IF IA32_EFER.LMA == 1
    THEN CR3 := R8;
EIP := tempEIP;
END;
```

Flags Affected

None.

Use of Prefixes

LOCK	Causes #UD.
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ).
Operand size	Causes #UD.
NP	66/F2/F3 prefixes are not allowed.
Segment overrides	Ignored.
Address size	Ignored.
REX.W	Sets 64-bit mode Operand size attribute.

Protected Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[EXITAC] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.PE = 0 or CPL > 0 or EFLAGS.VM = 1. If in VMX root operation. If the processor is not currently in authenticated code execution mode. If the processor is in SMM. If any reserved bit position is set in the EDX parameter register.

Real-Address Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[EXITAC] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[EXITAC] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[EXITAC] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[EXITAC] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

64-Bit Mode Exceptions

All protected mode exceptions apply.

#GP(0)	If the target address in RBX is not in a canonical form.

VM-Exit Condition

Reason (GETSEC)	If in VMX non-root operation.

GETSEC[SENTER]—Enter a Measured Environment

Opcode	Instruction	Description
NP OF 37 (EAX=4)	GETSEC[SENTER]	Launch a measured environment. EBX holds the SINIT authenticated code module physical base address. ECX holds the SINIT authenticated code module size (bytes). EDX controls the level of functionality supported by the measured environment launch.

Description

The GETSEC[SENTER] instruction initiates the launch of a measured environment and places the initiating logical processor (ILP) into the authenticated code execution mode. The SENTER leaf of GETSEC is selected with EAX set to 4 at execution. The physical base address of the AC module to be loaded and authenticated is specified in EBX. The size of the module in bytes is specified in ECX. EDX controls the level of functionality supported by the measured environment launch. To enable the full functionality of the protected environment launch, EDX must be initialized to zero.

The authenticated code base address and size parameters (in bytes) are passed to the GETSEC[SENTER] instruction using EBX and ECX respectively. The ILP evaluates the contents of these registers according to the rules for the AC module address in GETSEC[ENTERACCS]. AC module execution follows the same rules, as set by GETSEC[ENTERACCS].

The launching software must ensure that the TPM.ACCESS_0.activeLocality bit is clear before executing the GETSEC[SENTER] instruction.

There are restrictions enforced by the processor for execution of the GETSEC[SENTER] instruction:

- Execution is not allowed unless the processor is in protected mode or IA-32e mode with CPL = 0 and EFLAGS.VM = 0.
- Processor cache must be available and not disabled using the CR0.CD and NW bits.
- For enforcing consistency of operation with numeric exception reporting using Interrupt 16, CR0.NE must be set.
- An Intel TXT-capable chipset must be present as communicated to the processor by sampling of the power-on configuration capability field after reset.
- The processor can not be in authenticated code execution mode or already in a measured environment (as launched by a previous GETSEC[ENTERACCS] or GETSEC[SENTER] instruction).
- To avoid potential operability conflicts between modes, the processor is not allowed to execute this instruction if it currently is in SMM or VMX operation.
- To ensure consistent handling of SIPI messages, the processor executing the GETSEC[SENTER] instruction must also be designated the BSP (boot-strap processor) as defined by IA32_APIC_BASE.BSP (Bit 8).
- EDX must be initialized to a setting supportable by the processor. Unless enumeration by the GETSEC[PARAMETERS] leaf reports otherwise, only a value of zero is supported.

Failure to abide by the above conditions results in the processor signaling a general protection violation.

This instruction leaf starts the launch of a measured environment by initiating a rendezvous sequence for all logical processors in the platform. The rendezvous sequence involves the initiating logical processor sending a message (by executing GETSEC[SENTER]) and other responding logical processors (RLPs) acknowledging the message, thus synchronizing the RLP(s) with the ILP.

In response to a message signaling the completion of rendezvous, RLPs clear the bootstrap processor indicator flag (IA32_APIC_BASE.BSP) and enter an SENTER sleep state. In this sleep state, RLPs enter an idle processor condition while waiting to be activated after a measured environment has been established by the system executive. RLPs in the SENTER sleep state can only be activated by the GETSEC leaf function WAKEUP in a measured environment.

A successful launch of the measured environment results in the initiating logical processor entering the authenticated code execution mode. Prior to reaching this point, the ILP performs the following steps internally:

- Inhibit processor response to the external events: INIT, A20M, NMI, and SMI.
- Establish and check the location and size of the authenticated code module to be executed by the ILP.
- Check for the existence of an Intel® TXT-capable chipset.
- Verify the current power management configuration is acceptable.
- Broadcast a message to enable protection of memory and I/O from activities from other processor agents.
- Load the designated AC module into authenticated code execution area.
- Isolate the content of authenticated code execution area from further state modification by external agents.
- Authenticate the AC module.
- Updated the Trusted Platform Module (TPM) with the authenticated code module's hash.
- Initialize processor state based on the authenticated code module header information.
- Unlock the Intel® TXT-capable chipset private configuration register space and TPM locality 3 space.
- Begin execution in the authenticated code module at the defined entry point.

As an integrity check for proper processor hardware operation, execution of GETSEC[SENDER] will also check the contents of all the machine check status registers (as reported by the MSRs IA32_MCI_STATUS) for any valid uncorrectable error condition. In addition, the global machine check status register IA32_MCG_STATUS MCIP bit must be cleared and the IERR processor package pin (or its equivalent) must be not asserted, indicating that no machine check exception processing is currently in-progress. These checks are performed twice: once by the ILP prior to the broadcast of the rendezvous message to RLPs, and later in response to RLPs acknowledging the rendezvous message. Any outstanding valid uncorrectable machine check error condition present in the machine check status registers at the first check point will result in the ILP signaling a general protection violation. If an outstanding valid uncorrectable machine check error condition is present at the second check point, then this will result in the corresponding logical processor signaling the more severe TXT-shutdown condition with an error code of 12.

Before loading and authentication of the target code module is performed, the processor also checks that the current voltage and bus ratio encodings correspond to known good values supportable by the processor. The MSR IA32_PERF_STATUS values are compared against either the processor supported maximum operating target setting, system reset setting, or the thermal monitor operating target. If the current settings do not meet any of these criteria then the SENTER function will attempt to change the voltage and bus ratio select controls in a processor-specific manner. This adjustment may be to the thermal monitor, minimum (if different), or maximum operating target depending on the processor.

This implies that some thermal operating target parameters configured by BIOS may be overridden by SENTER. The measured environment software may need to take responsibility for restoring such settings that are deemed to be safe, but not necessarily recognized by SENTER. If an adjustment is not possible when an out of range setting is discovered, then the processor will abort the measured launch. This may be the case for chipset controlled settings of these values or if the controllability is not enabled on the processor. In this case it is the responsibility of the external software to program the chipset voltage ID and/or bus ratio select settings to known good values recognized by the processor, prior to executing SENTER.

NOTE

For a mobile processor, an adjustment can be made according to the thermal monitor operating target. For a quad-core processor the SENTER adjustment mechanism may result in a more conservative but non-uniform voltage setting, depending on the pre-SENDER settings per core.

The ILP and RLPs mask the response to the assertion of the external signals INIT#, A20M, NMI#, and SMI#. The purpose of this masking control is to prevent exposure to existing external event handlers until a protected handler has been put in place to directly handle these events. Masked external pin events may be unmasked conditionally or unconditionally via the GETSEC[EXITAC], GETSEC[SEXIT], GETSEC[SMCTRL] or for specific VMX related operations such as a VM entry or the VMXOFF instruction (see respective GETSEC leaves and Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C, for more details). The state of the A20M pin is masked and forced internally to a de-asserted state so that external assertion is not recognized. A20M masking as set by

GETSEC[SENDER] is undone only after taking down the measured environment with the GETSEC[SEXIT] instruction or processor reset. INTR is masked by simply clearing the EFLAGS.IF bit. It is the responsibility of system software to control the processor response to INTR through appropriate management of EFLAGS.

To prevent other (logical) processors from interfering with the ILP operating in authenticated code execution mode, memory (excluding implicit write-back transactions) and I/O activities originating from other processor agents are blocked. This protection starts when the ILP enters into authenticated code execution mode. Only memory and I/O transactions initiated from the ILP are allowed to proceed. Exiting authenticated code execution mode is done by executing GETSEC[EXITAC]. The protection of memory and I/O activities remains in effect until the ILP executes GETSEC[EXITAC].

Once the authenticated code module has been loaded into the authenticated code execution area, it is protected against further modification from external bus snoops. There is also a requirement that the memory type for the authenticated code module address range be WB (via initialization of the MTRRs prior to execution of this instruction). If this condition is not satisfied, it is a violation of security and the processor will force a TXT system reset (after writing an error code to the chipset LT.ERRORCODE register). This action is referred to as a Intel® TXT reset condition. It is performed when it is considered unreliable to signal an error through the conventional exception reporting mechanism.

To conform to the minimum granularity of MTRR MSRs for specifying the memory type, authenticated code RAM (ACRAM) is allocated to the processor in 4096 byte granular blocks. If an AC module size as specified in ECX is not a multiple of 4096 then the processor will allocate up to the next 4096 byte boundary for mapping as ACRAM with indeterminate data. This pad area will not be visible to the authenticated code module as external memory nor can it depend on the value of the data used to fill the pad area.

Once successful authentication has been completed by the ILP, the computed hash is stored in a trusted storage facility in the platform. The following trusted storage facilities are supported:

- If the platform register FTM_INTERFACE_ID.[bits 3:0] = 0, the computed hash is stored to the platform's TPM at PCR17 after this register is implicitly reset. PCR17 is a dedicated register for holding the computed hash of the authenticated code module loaded and subsequently executed by the GETSEC[SENDER]. As part of this process, the dynamic PCRs 18-22 are reset so they can be utilized by subsequently software for registration of code and data modules.
- If the platform register FTM_INTERFACE_ID.[bits 3:0] = 1, the computed hash is stored in a firmware trusted module (FTM) using a modified protocol similar to the protocol used to write to TPM's PCR17.

After successful execution of SENDER, either PCR17 (if FTM is not enabled) or the FTM (if enabled) contains the measurement of AC code and the SENDER launching parameters.

After authentication is completed successfully, the private configuration space of the Intel® TXT-capable chipset is unlocked so that the authenticated code module and measured environment software can gain access to this normally restricted chipset state. The Intel® TXT-capable chipset private configuration space can be locked later by software writing to the chipset LT.CMD.CLOSE-PRIVATE register or unconditionally using the GETSEC[SEXIT] instruction.

The SENTER leaf function also initializes some processor architecture state for the ILP from contents held in the header of the authenticated code module. Since the authenticated code module is relocatable, all address references are relative to the base address passed in via EBX. The ILP GDTR base value is initialized to EBX + [GDTBasePtr] and GDTR limit set to [GDTLimit]. The CS selector is initialized to the value held in the AC module header field SegSel, while the DS, SS, and ES selectors are initialized to CS+8. The segment descriptor fields are initialized implicitly with BASE=0, LIMIT=FFFFFh, G=1, D=1, P=1, S=1, read/write/accessed for DS, SS, and ES, while execute/read/accessed for CS. Execution in the authenticated code module for the ILP begins with the EIP set to EBX + [EntryPoint]. AC module defined fields used for initializing processor state are consistency checked with a failure resulting in an TXT-shutdown condition.

Table 7-6 provides a summary of processor state initialization for the ILP and RLP(s) after successful completion of GETSEC[SENDER]. For both ILP and RLP(s), paging is disabled upon entry to the measured environment. It is up to the ILP to establish a trusted paging environment, with appropriate mappings, to meet protection requirements established during the launch of the measured environment. RLP state initialization is not completed until a subsequent wake-up has been signaled by execution of the GETSEC[WAKEUP] function by the ILP.

Table 7-6. Register State Initialization After GETSEC[SENDER] and GETSEC[WAKEUP]

Register State	ILP after GETSEC[SENDER]	RLP after GETSEC[WAKEUP]
CR0	PG←0, AM←0, WP←0; Others unchanged	PG←0, CD←0, NW←0, AM←0, WP←0; PE←1, NE←1
CR4	00004000H	00004000H
EFLAGS	00000002H	00000002H
IA32_EFER	0H	0
EIP	[EntryPoint from MLE header ¹]	[LT.MLE.JOIN + 12]
EBX	Unchanged [SINIT.BASE]	Unchanged
EDX	SENDER control flags	Unchanged
EBP	SINIT.BASE	Unchanged
CS	Sel=[SINIT.SegSel], base=0, limit=FFFFFh, G=1, D=1, AR=9BH	Sel = [LT.MLE.JOIN + 8], base = 0, limit = FFFFFH, G = 1, D = 1, AR = 9BH
DS, ES, SS	Sel=[SINIT.SegSel] + 8, base=0, limit=FFFFFh, G=1, D=1, AR=93H	Sel = [LT.MLE.JOIN + 8] + 8, base = 0, limit = FFFFFH, G = 1, D = 1, AR = 93H
GDTR	Base= SINIT.base (EBX) + [SINIT.GDTBasePtr], Limit=[SINIT.GDTLimit]	Base = [LT.MLE.JOIN + 4], Limit = [LT.MLE.JOIN]
DR7	00000400H	00000400H
IA32_DEBUGCTL	0H	0H
Performance counters and counter control registers	0H	0H
IA32_MISC_ENABLE	See Table 7-5	See Table 7-5
IA32_SMM_MONITOR_CTL	Bit 2←0	Bit 2←0

NOTES:

1. See the *Intel® Trusted Execution Technology Measured Launched Environment Programming Guide* for MLE header format.

Segmentation related processor state that has not been initialized by GETSEC[SENDER] requires appropriate initialization before use. Since a new GDT context has been established, the previous state of the segment selector values held in FS, GS, TR, and LDTR may no longer be valid. The IDTR will also require reloading with a new IDT context after launching the measured environment before exceptions or the external interrupts INTR and NMI can be handled. In the meantime, the programmer must take care in not executing an INT n instruction or any other condition that would result in an exception or trap signaling.

Debug exception and trap related signaling is also disabled as part of execution of GETSEC[SENDER]. This is achieved by clearing DR7, TF in EFLAGS, and the MSR IA32_DEBUGCTL as defined in Table 7-6. These can be re-enabled once supporting exception handler(s), descriptor tables, and debug registers have been properly re-initialized following SENTER. Also, any pending single-step trap condition will be cleared at the completion of SENTER for both the ILP and RLP(s).

Performance related counters and counter control registers are cleared as part of execution of SENTER on both the ILP and RLP. This implies any active performance counters at the time of SENTER execution will be disabled. To reactive the processor performance counters, this state must be re-initialized and re-enabled.

Since MCE along with all other state bits (with the exception of SMXE) are cleared in CR4 upon execution of SENTER processing, any enabled machine check error condition that occurs will result in the processor performing the TXT-shutdown action. This also applies to an RLP while in the SENTER sleep state. For each logical processor CR4.MCE must be reestablished with a valid machine check exception handler to otherwise avoid an TXT-shutdown under such conditions.

The MSR IA32_EFER is also unconditionally cleared as part of the processor state initialized by SENTER for both the ILP and RLP. Since paging is disabled upon entering authenticated code execution mode, a new paging environment will have to be re-established if it is desired to enable IA-32e mode while operating in authenticated code execution mode.

The miscellaneous feature control MSR, IA32_MISC_ENABLE, is initialized as part of the measured environment launch. Certain bits of this MSR are preserved because preserving these bits may be important to maintain previously established platform settings. See the footnote for Table 7-5 The remaining bits are cleared for the purpose of establishing a more consistent environment for the execution of authenticated code modules. Among the impact of initializing this MSR, any previous condition established by the MONITOR instruction will be cleared.

Effect of MSR IA32_FEATURE_CONTROL MSR

Bits 15:8 of the IA32_FEATURE_CONTROL MSR affect the execution of GETSEC[SENTER]. These bits consist of two fields:

- Bit 15: a global enable control for execution of SENTER.
- Bits 14:8: a parameter control field providing the ability to qualify SENTER execution based on the level of functionality specified with corresponding EDX parameter bits 6:0.

The layout of these fields in the IA32_FEATURE_CONTROL MSR is shown in Table 7-1.

Prior to the execution of GETSEC[SENTER], the lock bit of IA32_FEATURE_CONTROL MSR must be bit set to affirm the settings to be used. Once the lock bit is set, only a power-up reset condition will clear this MSR. The IA32_FEATURE_CONTROL MSR must be configured in accordance to the intended usage at platform initialization. Note that this MSR is only available on SMX or VMX enabled processors. Otherwise, IA32_FEATURE_CONTROL is treated as reserved.

The Intel® Trusted Execution Technology Measured Launched Environment Programming Guide provides additional details and requirements for programming measured environment software to launch in an Intel TXT platform.

Operation in a Uni-Processor Platform

(* The state of the internal flag ACMODEFLAG and SENTERFLAG persist across instruction boundary *)

GETSEC[SENTER] (ILP Only):

```

IF (CR4.SMXE=0)
    THEN #UD;
ELSE IF (in VMX non-root operation)
    THEN VM Exit (reason="GETSEC instruction");
ELSE IF (GETSEC leaf unsupported)
    THEN #UD;
ELSE IF ((in VMX root operation) or
    (CR0.PE=0) or (CR0.CD=1) or (CR0.NW=1) or (CR0.NE=0) or
    (CPL>0) or (EFLAGS.VM=1) or
    (IA32_APIC_BASE.BSP=0) or (TXT chipset not present) or
    (SENTERFLAG=1) or (ACMODEFLAG=1) or (IN_SMM=1) or
    (TPM interface is not present) or
    (EDX ≠ (SENTER_EDX_support_mask & EDX)) or
    (IA32_FEATURE_CONTROL[0]=0) or (IA32_FEATURE_CONTROL[15]=0) or
    ((IA32_FEATURE_CONTROL[14:8] & EDX[6:0]) ≠ EDX[6:0]))
    THEN #GP(0);
IF (GETSEC[PARAMETERS].Parameter_Type = 5, MCA_Handling (bit 6) = 0)
    FOR I = 0 to IA32_MCG_CAP.COUNT-1 DO
        IF IA32_MCG[I].STATUS = uncorrectable error
            THEN #GP(0);
    FI;
OD;
FI;
IF (IA32_MCG_STATUS.MCIP=1) or (IERR pin is asserted)

```

```

    THEN #GP(0);
ACBASE := EBX;
ACSIZE := ECX;
IF (((ACBASE MOD 4096) ≠ 0) or ((ACSIZE MOD 64) ≠ 0) or (ACSIZE < minimum
    module size) or (ACSIZE > AC RAM capacity) or ((ACBASE+ACSIZE) > (232 - 1)))
    THEN #GP(0);
Mask SMI, INIT, A20M, and NMI external pin events;
SignalTXTMsg(SENTER);
DO
WHILE (no SignalSENTER message);

```

TXT_SENTER_MSG_EVENT (ILP & RLP):

```

Mask and clear SignalSENTER event;
Unmask SignalSEXIT event;
IF (in VMX operation)
    THEN TXT-SHUTDOWN(#IllegalEvent);
FOR I = 0 to IA32_MCG_CAP.COUNT-1 DO
    IF IA32_MCG[I].STATUS = uncorrectable error
        THEN TXT-SHUTDOWN(#UnrecovMCErr);
    FI;
OD;
IF (IA32_MCG_STATUS.MCIP=1) or (IERR pin is asserted)
    THEN TXT-SHUTDOWN(#UnrecovMCErr);
IF (Voltage or bus ratio status are NOT at a known good state)
    THEN IF (Voltage select and bus ratio are internally adjustable)
        THEN
            Make product-specific adjustment on operating parameters;
        ELSE
            TXT-SHUTDOWN(#IllegalVIDBRatio);
    FI;
FI;

IA32_MISC_ENABLE := (IA32_MISC_ENABLE & MASK_CONST*)
(* The hexadecimal value of MASK_CONST may vary due to processor implementations *)
A20M := 0;
IA32_DEBUGCTL := 0;
Invalidate processor TLB(s);
Drain outgoing transactions;
Clear performance monitor counters and control;
SENTERFLAG := 1;
SignalTXTMsg(SENTERAck);
IF (logical processor is not ILP)
    THEN GOTO RLP_SENTER_ROUTINE;
(* ILP waits for all logical processors to ACK *)
DO
    DONE := TXT.READ(LT.STS);
WHILE (not DONE);
SignalTXTMsg(SENTERContinue);
SignalTXTMsg(ProcessorHold);
FOR I=ACBASE to ACRAM+ACSIZE-1 DO
    ACRAM[I-ACBASE].ADDR := I;
    ACRAM[I-ACBASE].DATA := LOAD(I);
OD;
IF (ACRAM memory type ≠ WB)
    THEN TXT-SHUTDOWN(#BadACMMType);

```

```

IF (AC module header version is not supported) OR (ACRAM[ModuleType] ≠ 2)
    THEN TXT-SHUTDOWN(#UnsupportedACM);
KEY := GETKEY(ACRAM, ACBASE);
KEYHASH := HASH(KEY);
CSKEYHASH := LT.READ(LT.PUBLIC.KEY);
IF (KEYHASH ≠ CSKEYHASH)
    THEN TXT-SHUTDOWN(#AuthenticateFail);
SIGNATURE := DECRYPT(ACRAM, ACBASE, KEY);
(* The value of SIGNATURE_LEN_CONST is implementation-specific*)
FOR I=0 to SIGNATURE_LEN_CONST - 1 DO
    ACRAM[SCRATCH.I] := SIGNATURE[I];
COMPUTEDSIGNATURE := HASH(ACRAM, ACBASE, ACSIZE);
FOR I=0 to SIGNATURE_LEN_CONST - 1 DO
    ACRAM[SCRATCH.SIGNATURE_LEN_CONST+I] := COMPUTEDSIGNATURE[I];
IF (SIGNATURE ≠ COMPUTEDSIGNATURE)
    THEN TXT-SHUTDOWN(#AuthenticateFail);
ACMCONTROL := ACRAM[CodeControl];
IF ((ACMCONTROL.0 = 0) and (ACMCONTROL.1 = 1) and (snoop hit to modified line detected on ACRAM load))
    THEN TXT-SHUTDOWN(#UnexpectedHITM);
IF (ACMCONTROL reserved bits are set)
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACRAM[GDTBasePtr] < (ACRAM[HeaderLen] * 4 + Scratch_size)) OR
    ((ACRAM[GDTBasePtr] + ACRAM[GDTLimit]) >= ACSIZE))
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACMCONTROL.0 = 1) and (ACMCONTROL.1 = 1) and (snoop hit to modified
    line detected on ACRAM load))
    THEN ACEntryPoint := ACBASE+ACRAM[ErrorEntryPoint];
ELSE
    ACEntryPoint := ACBASE+ACRAM[EntryPoint];
IF ((ACEntryPoint >= ACSIZE) or (ACEntryPoint < (ACRAM[HeaderLen] * 4 + Scratch_size)))
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACRAM[SegSel] > (ACRAM[GDTLimit] - 15)) or (ACRAM[SegSel] < 8))
    THEN TXT-SHUTDOWN(#BadACMFormat);
IF ((ACRAM[SegSel].TI=1) or (ACRAM[SegSel].RPL≠0))
    THEN TXT-SHUTDOWN(#BadACMFormat);

IF (FTM_INTERFACE_ID.[3:0] = 1 ) (* Alternate FTM Interface has been enabled *)
    THEN (* TPM_LOC_CTRL_4 is located at 0FED44008H, TMP_DATA_BUFFER_4 is located at 0FED44080H *)
        WRITE(TPM_LOC_CTRL_4) := 01H; (* Modified HASH.START protocol *)
        (* Write to firmware storage *)
        WRITE(TPM_DATA_BUFFER_4) := SIGNATURE_LEN_CONST + 4;
        FOR I=0 to SIGNATURE_LEN_CONST - 1 DO
            WRITE(TPM_DATA_BUFFER_4 + 2 + I) := ACRAM[SCRATCH.I];
            WRITE(TPM_DATA_BUFFER_4 + 2 + SIGNATURE_LEN_CONST) := EDX;
        WRITE(FTM.LOC_CTRL) := 06H; (* Modified protocol combining HASH.DATA and HASH.END *)
    ELSE IF (FTM_INTERFACE_ID.[3:0] = 0 ) (* Use standard TPM Interface *)
        ACRAM[SCRATCH.SIGNATURE_LEN_CONST] := EDX;
        WRITE(TPM.HASH.START) := 0;
        FOR I=0 to SIGNATURE_LEN_CONST + 3 DO
            WRITE(TPM.HASH.DATA) := ACRAM[SCRATCH.I];
            WRITE(TPM.HASH.END) := 0;
FI;
ACMODEFLAG := 1;
CRO.[PG.AM.WP] := 0;

```

```

CR4 := 00004000h;
EFLAGS := 00000002h;
IA32_EFER := 0;
EBP := ACBASE;
GDTR.BASE := ACBASE+ACRAM[GDTBasePtr];
GDTR.LIMIT := ACRAM[GDTLimit];
CS.SEL := ACRAM[SegSel];
CS.BASE := 0;
CS.LIMIT := FFFFFFFh;
CS.G := 1;
CS.D := 1;
CS.AR := 9Bh;
DS.SEL := ACRAM[SegSel]+8;
DS.BASE := 0;
DS.LIMIT := FFFFFFFh;
DS.G := 1;
DS.D := 1;
DS.AR := 93h;
SS := DS;
ES := DS;
DR7 := 00000400h;
IA32_DEBUGCTL := 0;
SignalTXTMsg(UnlockSMRAM);
SignalTXTMsg(OpenPrivate);
SignalTXTMsg(OpenLocality3);
EIP := ACEntryPoint;
END;

```

RLP_SENTER_ROUTINE: (RLP Only)

```

Mask SMI, INIT, A20M, and NMI external pin events
Unmask SignalWAKEUP event;
Wait for SignalSENTERContinue message;
IA32_APIC_BASE.BSP := 0;
GOTO SENTER sleep state;
END;

```

Flags Affected

All flags are cleared.

Use of Prefixes

LOCK	Causes #UD.
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ).
Operand size	Causes #UD.
NP	66/F2/F3 prefixes are not allowed.
Segment overrides	Ignored.
Address size	Ignored.
REX	Ignored.

Protected Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[SENDER] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.CD = 1 or CR0.NW = 1 or CR0.NE = 0 or CR0.PE = 0 or CPL > 0 or EFLAGS.VM = 1. If in VMX root operation. If the initiating processor is not designated as the bootstrap processor via the MSR bit IA32_APIC_BASE.BSP. If an Intel® TXT-capable chipset is not present. If an Intel® TXT-capable chipset interface to TPM is not detected as present. If a protected partition is already active or the processor is already in authenticated code mode. If the processor is in SMM. If a valid uncorrectable machine check error is logged in IA32_MC[I]_STATUS. If the authenticated code base is not on a 4096 byte boundary. If the authenticated code size > processor's authenticated code execution area storage capacity. If the authenticated code size is not modulo 64.

Real-Address Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[SENDER] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[SENDER] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[SENDER] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[SENDER] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

#GP	If AC code module does not reside in physical address below $2^{32} - 1$.

64-Bit Mode Exceptions

All protected mode exceptions apply.

#GP	If AC code module does not reside in physical address below $2^{32} - 1$.

VM-Exit Condition

Reason (GETSEC)	If in VMX non-root operation.

GETSEC[SEXIT]—Exit Measured Environment

Opcode	Instruction	Description
NP OF 37 (EAX=5)	GETSEC[SEXIT]	Exit measured environment.

Description

The GETSEC[SEXIT] instruction initiates an exit of a measured environment established by GETSEC[SENDER]. The SEXIT leaf of GETSEC is selected with EAX set to 5 at execution. This instruction leaf sends a message to all logical processors in the platform to signal the measured environment exit.

There are restrictions enforced by the processor for the execution of the GETSEC[SEXIT] instruction:

- Execution is not allowed unless the processor is in protected mode (CR0.PE = 1) with CPL = 0 and EFLAGS.VM = 0.
- The processor must be in a measured environment as launched by a previous GETSEC[SENDER] instruction, but not still in authenticated code execution mode.
- To avoid potential interoperability conflicts between modes, the processor is not allowed to execute this instruction if it currently is in SMM or in VMX operation.
- To ensure consistent handling of SIPI messages, the processor executing the GETSEC[SEXIT] instruction must also be designated the BSP (bootstrap processor) as defined by the register bit IA32_APIC_BASE.BSP (bit 8).

Failure to abide by the above conditions results in the processor signaling a general protection violation.

This instruction initiates a sequence to rendezvous the RLPs with the ILP. It then clears the internal processor flag indicating the processor is operating in a measured environment.

In response to a message signaling the completion of rendezvous, all RLPs restart execution with the instruction that was to be executed at the time GETSEC[SEXIT] was recognized. This applies to all processor conditions, with the following exceptions:

- If an RLP executed HLT and was in this halt state at the time of the message initiated by GETSEC[SEXIT], then execution resumes in the halt state.
- If an RLP was executing MWAIT, then a message initiated by GETSEC[SEXIT] causes an exit of the MWAIT state, falling through to the next instruction.
- If an RLP was executing an intermediate iteration of a string instruction, then the processor resumes execution of the string instruction at the point which the message initiated by GETSEC[SEXIT] was recognized.
- If an RLP is still in the SENTER sleep state (never awakened with GETSEC[WAKEUP]), it will be sent to the wait-for-SIPI state after first clearing the bootstrap processor indicator flag (IA32_APIC_BASE.BSP) and any pending SIPI state. In this case, such RLPs are initialized to an architectural state consistent with having taken a soft reset using the INIT# pin.

Prior to completion of the GETSEC[SEXIT] operation, both the ILP and any active RLPs unmask the response of the external event signals INIT#, A20M, NMI#, and SMI#. This unmasking is performed unconditionally to recognize pin events which are masked after a GETSEC[SENDER]. The state of A20M is unmasked, as the A20M pin is not recognized while the measured environment is active.

On a successful exit of the measured environment, the ILP re-locks the Intel® TXT-capable chipset private configuration space. GETSEC[SEXIT] does not affect the content of any PCR.

At completion of GETSEC[SEXIT] by the ILP, execution proceeds to the next instruction. Since EFLAGS and the debug register state are not modified by this instruction, a pending trap condition is free to be signaled if previously enabled.

Operation in a Uni-Processor Platform

(* The state of the internal flag ACMODEFLAG and SENTERFLAG persist across instruction boundary *)

GETSEC[SEXIT] (ILP Only):

```
IF (CR4.SMXE=0)
    THEN #UD;
ELSE IF (in VMX non-root operation)
    THEN VM Exit (reason="GETSEC instruction");
ELSE IF (GETSEC leaf unsupported)
    THEN #UD;
ELSE IF ((in VMX root operation) or
    (CR0.PE=0) or (CPL>0) or (EFLAGS.VM=1) or
    (IA32_APIC_BASE.BSP=0) or
    (TXT chipset not present) or
    (SENTERFLAG=0) or (ACMODEFLAG=1) or (IN_SMM=1))
    THEN #GP(0);
SignalTXTMsg(SEXIT);
DO
    WHILE (no SignalSEXIT message);
```

TXT_SEXIT_MSG_EVENT (ILP & RLP):

```
Mask and clear SignalSEXIT event;
Clear MONITOR FSM;
Unmask SignalSENDER event;
IF (in VMX operation)
    THEN TXT-SHUTDOWN(#IllegalEvent);
SignalTXTMsg(SEXITAck);
IF (logical processor is not ILP)
    THEN GOTO RLP_SEXIT_ROUTINE;
(* ILP waits for all logical processors to ACK *)
DO
    DONE := READ(LT.STS);
    WHILE (NOT DONE);
SignalTXTMsg(SEXITContinue);
SignalTXTMsg(ClosePrivate);
SENTERFLAG := 0;
Unmask SMI, INIT, A20M, and NMI external pin events;
END;
```

RLP_SEXIT_ROUTINE (RLPs Only):

```
Wait for SignalSEXITContinue message;
Unmask SMI, INIT, A20M, and NMI external pin events;
IF (prior execution state = HLT)
    THEN reenter HLT state;
IF (prior execution state = SENTER sleep)
    THEN
        IA32_APIC_BASE.BSP := 0;
        Clear pending SIPI state;
        Call INIT_PROCESSOR_STATE;
        Unmask SIPI event;
        GOTO WAIT-FOR-SIPI;
FI;
END;
```

Flags Affected

ILP: None.

RLPs: All flags are modified for an RLP. returning to wait-for-SIPI state, none otherwise.

Use of Prefixes

LOCK	Causes #UD.
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ).
Operand size	Causes #UD.
NP	66/F2/F3 prefixes are not allowed.
Segment overrides	Ignored.
Address size	Ignored.
REX	Ignored.

Protected Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[SEXIT] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.PE = 0 or CPL > 0 or EFLAGS.VM = 1. If in VMX root operation. If the initiating processor is not designated via the MSR bit IA32_APIC_BASE.BSP. If an Intel® TXT-capable chipset is not present. If a protected partition is not already active or the processor is already in authenticated code mode. If the processor is in SMM.

Real-Address Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[SEXIT] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[SEXIT] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[SEXIT] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[SEXIT] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

64-Bit Mode Exceptions

All protected mode exceptions apply.

VM-Exit Condition

Reason (GETSEC) If in VMX non-root operation.

GETSEC[PARAMETERS]—Report the SMX Parameters

Opcode	Instruction	Description
NP OF 37 (EAX=6)	GETSEC[PARAMETERS]	Report the SMX parameters. The parameters index is input in EBX with the result returned in EAX, EBX, and ECX.

Description

The GETSEC[PARAMETERS] instruction returns specific parameter information for SMX features supported by the processor. Parameter information is returned in EAX, EBX, and ECX, with the input parameter selected using EBX.

Software retrieves parameter information by searching with an input index for EBX starting at 0, and then reading the returned results in EAX, EBX, and ECX. EAX[4:0] is designated to return a parameter type field indicating if a parameter is available and what type it is. If EAX[4:0] is returned with 0, this designates a null parameter and indicates no more parameters are available.

Table 7-7 defines the parameter types supported in current and future implementations.

Table 7-7. SMX Reporting Parameters Format

Parameter Type EAX[4:0]	Parameter Description	EAX[31:5]	EBX[31:0]	ECX[31:0]
0	NULL	Reserved (0 returned)	Reserved (unmodified)	Reserved (unmodified)
1	Supported AC module versions	Reserved (0 returned)	Version comparison mask	Version numbers supported
2	Max size of authenticated code execution area	Multiply by 32 for size in bytes	Reserved (unmodified)	Reserved (unmodified)
3	External memory types supported during AC mode	Memory type bit mask	Reserved (unmodified)	Reserved (unmodified)
4	Selective SENTER functionality control	EAX[14:8] correspond to available SENTER function disable controls	Reserved (unmodified)	Reserved (unmodified)
5	TXT extensions support	TXT Feature Extensions Flags (see Table)	Reserved	Reserved
6-31	Undefined	Reserved (unmodified)	Reserved (unmodified)	Reserved (unmodified)

Table 7-8. TXT Feature Extensions Flags

Bit	Definition	Description
5	Processor based S-CRTM support	Returns 1 if this processor implements a processor-rooted S-CRTM capability and 0 if not (S-CRTM is rooted in BIOS). This flag cannot be used to infer whether the chipset supports TXT or whether the processor support SMX.
6	Machine Check Handling	Returns 1 if it machine check status registers can be preserved through ENTERACCS and SENTER. If this bit is 1, the caller of ENTERACCS and SENTER is not required to clear machine check error status bits before invoking these GETSEC leaves. If this bit returns 0, the caller of ENTERACCS and SENTER must clear all machine check error status bits before invoking these GETSEC leaves.
31:7	Reserved	Reserved for future use. Will return 0.

Supported AC module versions (as defined by the AC module HeaderVersion field) can be determined for a particular SMX capable processor by the type 1 parameter. Using EBX to index through the available parameters reported by GETSEC[PARAMETERS] for each unique parameter set returned for type 1, software can determine the complete list of AC module version(s) supported.

For each parameter set, EBX returns the comparison mask and ECX returns the available HeaderVersion field values supported, after AND'ing the target HeaderVersion with the comparison mask. Software can then determine if a particular AC module version is supported by following the pseudo-code search routine given below:

```
parameter_search_index= 0
do {
    EBX= parameter_search_index++
    EAX= 6
    GETSEC
    if (EAX[4:0] = 1) {
        if ((version_query & EBX) = ECX) {
            version_is_supported= 1
            break
        }
    }
} while (EAX[4:0] ≠ 0)
```

If only AC modules with a HeaderVersion of 0 are supported by the processor, then only one parameter set of type 1 will be returned, as follows: EAX = 00000001H,

EBX = FFFFFFFFH and ECX = 00000000H.

The maximum capacity for an authenticated code execution area supported by the processor is reported with the parameter type of 2. The maximum supported size in bytes is determined by multiplying the returned size in EAX[31:5] by 32. Thus, for a maximum supported authenticated RAM size of 32KBytes, EAX returns with 00008002H.

Supportable memory types for memory mapped outside of the authenticated code execution area are reported with the parameter type of 3. While is active, as initiated by the GETSEC functions SENTER and ENTERACCS and terminated by EXITAC, there are restrictions on what memory types are allowed for the rest of system memory. It is the responsibility of the system software to initialize the memory type range register (MTRR) MSRs and/or the page attribute table (PAT) to only map memory types consistent with the reporting of this parameter. The reporting of supportable memory types of external memory is indicated using a bit map returned in EAX[31:8]. These bit positions correspond to the memory type encodings defined for the MTRR MSR and PAT programming. See Table 7-9.

The parameter type of 4 is used for enumerating the availability of selective GETSEC[SENDER] function disable controls. If a 1 is reported in bits 14:8 of the returned parameter EAX, then this indicates a disable control capability exists with SENTER for a particular function. The enumerated field in bits 14:8 corresponds to use of the EDX input parameter bits 6:0 for SENTER. If an enumerated field bit is set to 1, then the corresponding EDX input parameter bit of EDX may be set to 1 to disable that designated function. If the enumerated field bit is 0 or this parameter is not reported, then no disable capability exists with the corresponding EDX input parameter for SENTER, and EDX bit(s) must be cleared to 0 to enable execution of SENTER. If no selective disable capability for SENTER exists as enumerated, then the corresponding bits in the IA32_FEATURE_CONTROL MSR bits 14:8 must also be programmed to 1 if the SENTER global enable bit 15 of the MSR is set. This is required to enable future extensibility of SENTER selective disable capability with respect to potentially separate software initialization of the MSR.

Table 7-9. External Memory Types Using Parameter 3

EAX Bit Position	Parameter Description
8	Uncacheable (UC)
9	Write Combining (WC)
11:10	Reserved
12	Write-through (WT)

Table 7-9. External Memory Types Using Parameter 3 (Contd.)

13	Write-protected (WP)
14	Write-back (WB)
31:15	Reserved

If the GETSEC[PARAMETERS] leaf or specific parameter is not present for a given SMX capable processor, then default parameter values should be assumed. These are defined in Table 7-10.

Table 7-10. Default Parameter Values

Parameter Type EAX[4:0]	Default Setting	Parameter Description
1	0.0 only	Supported AC module versions.
2	32 KBytes	Authenticated code execution area size.
3	UC only	External memory types supported during AC execution mode.
4	None	Available SENTER selective disable controls.

Operation

(* example of a processor supporting only a 0.0 HeaderVersion, 32K ACRAM size, memory types UC and WC *)

IF (CR4.SMXE=0)

THEN #UD;

ELSE IF (in VMX non-root operation)

THEN VM Exit (reason="GETSEC instruction");

ELSE IF (GETSEC leaf unsupported)

THEN #UD;

(* example of a processor supporting a 0.0 HeaderVersion *)

IF (EBX=0) THEN

EAX := 00000001h;

EBX := FFFFFFFFh;

ECX := 00000000h;

ELSE IF (EBX=1)

(* example of a processor supporting a 32K ACRAM size *)

THEN EAX := 00008002h;

ESE IF (EBX= 2)

(* example of a processor supporting external memory types of UC and WC *)

THEN EAX := 00000303h;

ESE IF (EBX= other value(s) less than unsupported index value)

(* EAX value varies. Consult Table 7-7 and Table *)

ELSE (* unsupported index*)

EAX := 00000000h;

END;

Flags Affected

None.

Use of Prefixes

LOCK Causes #UD.

REP* Cause #UD (includes REPNE/REPZ and REP/REPE/REPZ).

Operand size Causes #UD.

NP	66/F2/F3 prefixes are not allowed.
Segment overrides	Ignored.
Address size	Ignored.
REX	Ignored.

Protected Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[PARAMETERS] is not reported as supported by GETSEC[CAPABILITIES].

Real-Address Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[PARAMETERS] is not reported as supported by GETSEC[CAPABILITIES].

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[PARAMETERS] is not reported as supported by GETSEC[CAPABILITIES].

Compatibility Mode Exceptions

All protected mode exceptions apply.

64-Bit Mode Exceptions

All protected mode exceptions apply.

VM-Exit Condition

Reason (GETSEC)	If in VMX non-root operation.

GETSEC[SMCTRL]—SMX Mode Control

Opcode	Instruction	Description
NP OF 37 (EAX = 7)	GETSEC[SMCTRL]	Perform specified SMX mode control as selected with the input EBX.

Description

The GETSEC[SMCTRL] instruction is available for performing certain SMX specific mode control operations. The operation to be performed is selected through the input register EBX. Currently only an input value in EBX of 0 is supported. All other EBX settings will result in the signaling of a general protection violation.

If EBX is set to 0, then the SMCTRL leaf is used to re-enable SMI events. SMI is masked by the ILP executing the GETSEC[SENTER] instruction (SMI is also masked in the responding logical processors in response to SENTER rendezvous messages.). The determination of when this instruction is allowed and the events that are unmasked is dependent on the processor context (See Table 7-11). For brevity, the usage of SMCTRL where EBX=0 will be referred to as GETSEC[SMCTRL(0)].

As part of support for launching a measured environment, the SMI, NMI, and INIT events are masked after GETSEC[SENTER], and remain masked after exiting authenticated execution mode. Unmasking these events should be accompanied by securely enabling these event handlers. These security concerns can be addressed in VMX operation by a MVMM.

The VM monitor can choose two approaches:

- In a dual monitor approach, the executive software will set up an SMM monitor in parallel to the executive VMM (i.e., the MVMM), see Chapter 34, “System Management Mode,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C. The SMM monitor is dedicated to handling SMI events without compromising the security of the MVMM. This usage model of handling SMI while a measured environment is active does not require the use of GETSEC[SMCTRL(0)] as event re-enabling after the VMX environment launch is handled implicitly and through separate VMX based controls.
- If a dedicated SMM monitor will not be established and SMIs are to be handled within the measured environment, then GETSEC[SMCTRL(0)] can be used by the executive software to re-enable SMI that has been masked as a result of SENTER.

Table 7-11 defines the processor context in which GETSEC[SMCTRL(0)] can be used and which events will be unmasked. Note that the events that are unmasked are dependent upon the currently operating processor context.

Table 7-11. Supported Actions for GETSEC[SMCTRL(0)]

ILP Mode of Operation	SMCTRL execution action
In VMX non-root operation	VM exit
SENTERFLAG = 0	#GP(0), illegal context
In authenticated code execution mode (ACMODEFLAG = 1)	#GP(0), illegal context
SENTERFLAG = 1, not in VMX operation, not in SMM	Unmask SMI
SENTERFLAG = 1, in VMX root operation, not in SMM	Unmask SMI if SMM monitor is not configured, otherwise #GP(0)
SENTERFLAG = 1, In VMX root operation, in SMM	#GP(0), illegal context

Operation

(* The state of the internal flag ACMODEFLAG and SENTERFLAG persist across instruction boundary *)

```
IF (CR4.SMXE=0)
    THEN #UD;
ELSE IF (in VMX non-root operation)
    THEN VM Exit (reason="GETSEC instruction");
ELSE IF (GETSEC leaf unsupported)
    THEN #UD;
ELSE IF ((CR0.PE=0) or (CPL>0) OR (EFLAGS.VM=1))
    THEN #GP(0);
ELSE IF ((EBX=0) and (SENTERFLAG=1) and (ACMODEFLAG=0) and (IN_SMM=0) and
    (((in VMX root operation) and (SMM monitor not configured)) or (not in VMX operation)))
    THEN unmask SMI;
ELSE
    #GP(0);
END
```

Flags Affected

None.

Use of Prefixes

LOCK	Causes #UD.
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ).
Operand size	Causes #UD.
NP	66/F2/F3 prefixes are not allowed.
Segment overrides	Ignored.
Address size	Ignored.
REX	Ignored.

Protected Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[SMCTRL] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.PE = 0 or CPL > 0 or EFLAGS.VM = 1. If in VMX root operation. If a protected partition is not already active or the processor is currently in authenticated code mode. If the processor is in SMM. If the SMM monitor is not configured.

Real-Address Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[SMCTRL] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[SMCTRL] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[SMCTRL] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[SMCTRL] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

64-Bit Mode Exceptions

All protected mode exceptions apply.

VM-exit Condition

Reason (GETSEC) If in VMX non-root operation.

GETSEC[WAKEUP]—Wake Up Sleeping Processors in Measured Environment

Opcode	Instruction	Description
NP OF 37 (EAX=8)	GETSEC[WAKEUP]	Wake up the responding logical processors from the SENTER sleep state.

Description

The GETSEC[WAKEUP] leaf function broadcasts a wake-up message to all logical processors currently in the SENTER sleep state. This GETSEC leaf must be executed only by the ILP, in order to wake-up the RLPs. Responding logical processors (RLPs) enter the SENTER sleep state after completion of the SENTER rendezvous sequence.

The GETSEC[WAKEUP] instruction may only be executed:

- In a measured environment as initiated by execution of GETSEC[SENTER].
- Outside of authenticated code execution mode.
- Execution is not allowed unless the processor is in protected mode with CPL = 0 and EFLAGS.VM = 0.
- In addition, the logical processor must be designated as the boot-strap processor as configured by setting IA32_APIC_BASE.BSP = 1.

If these conditions are not met, attempts to execute GETSEC[WAKEUP] result in a general protection violation.

An RLP exits the SENTER sleep state and start execution in response to a WAKEUP signal initiated by ILP's execution of GETSEC[WAKEUP]. The RLP retrieves a pointer to a data structure that contains information to enable execution from a defined entry point. This data structure is located using a physical address held in the Intel® TXT-capable chipset configuration register LT.MLE.JOIN. The register is publicly writable in the chipset by all processors and is not restricted by the Intel® TXT-capable chipset configuration register lock status. The format of this data structure is defined in Table 7-12.

Table 7-12. RLP MVMM JOIN Data Structure

Offset	Field
0	GDT limit
4	GDT base pointer
8	Segment selector initializer
12	EIP

The MLE JOIN data structure contains the information necessary to initialize RLP processor state and permit the processor to join the measured environment. The GDTR, LIP, and CS, DS, SS, and ES selector values are initialized using this data structure. The CS selector index is derived directly from the segment selector initializer field; DS, SS, and ES selectors are initialized to CS+8. The segment descriptor fields are initialized implicitly with BASE = 0, LIMIT = FFFFH, G = 1, D = 1, P = 1, S = 1; read/write/access for DS, SS, and ES; and execute/read/access for CS. It is the responsibility of external software to establish a GDT pointed to by the MLE JOIN data structure that contains descriptor entries consistent with the implicit settings initialized by the processor (see Table 7-6). Certain states from the content of Table 7-12 are checked for consistency by the processor prior to execution. A failure of any consistency check results in the RLP aborting entry into the protected environment and signaling an Intel® TXT shutdown condition. The specific checks performed are documented later in this section. After successful completion of processor consistency checks and subsequent initialization, RLP execution in the measured environment begins from the entry point at offset 12 (as indicated in Table 7-12).

Operation

(* The state of the internal flag ACMODEFLAG and SENTERFLAG persist across instruction boundary *)

```
IF (CR4.SMXE=0)
    THEN #UD;
ELSE IF (in VMX non-root operation)
    THEN VM Exit (reason="GETSEC instruction");
ELSE IF (GETSEC leaf unsupported)
    THEN #UD;
ELSE IF ((CR0.PE=0) or (CPL>0) or (EFLAGS.VM=1) or (SENTERFLAG=0) or (ACMODEFLAG=1) or (IN_SMM=0) or (in VMX operation) or
(IA32_APIC_BASE.BSP=0) or (TXT chipset not present))
    THEN #GP(0);
ELSE
    SignalTXTMsg(WAKEUP);
END;
```

RLP_SIPWAKEUP_FROM_SENTER_ROUTINE: (RLP Only)

```
WHILE (no SignalWAKEUP event);
IF (IA32_SMM_MONITOR_CTL[0] ≠ ILP.IA32_SMM_MONITOR_CTL[0])
    THEN TXT-SHUTDOWN(#IllegalEvent)
IF (IA32_SMM_MONITOR_CTL[0] = 0)
    THEN Unmask SMI pin event;
ELSE
    Mask SMI pin event;
Mask A20M, and NMI external pin events (unmask INIT);
Mask SignalWAKEUP event;
Invalidate processor TLB(s);
Drain outgoing transactions;
TempGDTRLIMIT := LOAD(LT.MLE.JOIN);
TempGDTRBASE := LOAD(LT.MLE.JOIN+4);
TempSegSel := LOAD(LT.MLE.JOIN+8);
TempEIP := LOAD(LT.MLE.JOIN+12);
IF (TempGDTRLimit & FFFF0000h)
    THEN TXT-SHUTDOWN(#BadJOINFormat);
IF ((TempSegSel > TempGDTRLIMIT-15) or (TempSegSel < 8))
    THEN TXT-SHUTDOWN(#BadJOINFormat);
IF ((TempSegSel.TI=1) or (TempSegSel.RPL ≠ 0))
    THEN TXT-SHUTDOWN(#BadJOINFormat);
CR0.[PG,CD,NW,AM,WP] := 0;
CR0.[NE,PE] := 1;
CR4 := 00004000h;
EFLAGS := 00000002h;
IA32_EFER := 0;
GDTR.BASE := TempGDTRBASE;
GDTR.LIMIT := TempGDTRLIMIT;
CS.SEL := TempSegSel;
CS.BASE := 0;
CS.LIMIT := FFFFFh;
CS.G := 1;
CS.D := 1;
CS.AR := 9Bh;
DS.SEL := TempSegSel+8;
DS.BASE := 0;
DS.LIMIT := FFFFFh;
DS.G := 1;
```

```

DS.D := 1;
DS.AR := 93h;
SS := DS;
ES := DS;
DR7 := 00000400h;
IA32_DEBUGCTL := 0;
EIP := TempEIP;
END;

```

Flags Affected

None.

Use of Prefixes

LOCK	Causes #UD.
REP*	Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ).
Operand size	Causes #UD.
NP	66/F2/F3 prefixes are not allowed.
Segment overrides	Ignored.
Address size	Ignored.
REX	Ignored.

Protected Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[WAKEUP] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.PE = 0 or CPL > 0 or EFLAGS.VM = 1. If in VMX operation. If a protected partition is not already active or the processor is currently in authenticated code mode. If the processor is in SMM.
#UD	If CR4.SMXE = 0. If GETSEC[WAKEUP] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[WAKEUP] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0. If GETSEC[WAKEUP] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[WAKEUP] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

64-Bit Mode Exceptions

All protected mode exceptions apply.

VM-exit Condition

Reason (GETSEC) If in VMX non-root operation.

10. Updates to Appendix A, Volume 2D

Change bars and text show changes to Appendix A of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D: Instruction Set Reference, W-Z*.

Changes to this chapter:

- In Section A.3, "One, Two, and THREE-Byte Opcode Maps", updated Table A-3 to include CLDEMOT and WBNOINVD.
- In Section A.4, "Opcode Extensions For One-Byte And Two-byte Opcodes", updated Table A-6 adding the following:
 - CLFLUSHOPT
 - CLRSSBSY
 - CLUI
 - CLWB
 - ERETS
 - ERETU
 - INCSSP
 - LKGS
 - PBNKDB
 - PCONFIG
 - PTWRITE
 - RDMSRLIST
 - RDPKRU
 - RSTORSSP
 - SAVEPREVSSP
 - SAVES
 - SEAMCALL
 - SEAMOPS
 - SEAMRET
 - SENDUIPI
 - SERIALIZE
 - SETSSBSY
 - STUI
 - TDCALL
 - TPAUSE
 - UIRET
 - UMONITOR
 - UMWAIT
 - WRMSRLIST
 - WRMSRNS
 - WRPKRU
 - XRESLDTRK
 - XSAVEC
 - XSTORS
 - XSUSLDTRK

Use the opcode tables in this chapter to interpret IA-32 and Intel 64 architecture object code. Instructions are divided into encoding groups:

- 1-byte, 2-byte and 3-byte opcode encodings are used to encode integer, system, MMX technology, SSE/SSE2/SSE3/SSSE3/SSE4, and VMX instructions. Maps for these instructions are given in Table A-2 through Table A-6.
- Escape opcodes (in the format: ESC character, opcode, ModR/M byte) are used for floating-point instructions. The maps for these instructions are provided in Table A-7 through Table A-22.

NOTE

All blanks in opcode maps are reserved and must not be used. Do not depend on the operation of undefined or blank opcodes.

A.1 USING OPCODE TABLES

Tables in this appendix list opcodes of instructions (including required instruction prefixes, opcode extensions in associated ModR/M byte). Blank cells in the tables indicate opcodes that are reserved or undefined. Cells marked "Reserved-NOP" are also reserved but may behave as NOP on certain processors. Software should not use opcodes corresponding blank cells or cells marked "Reserved-NOP" nor depend on the current behavior of those opcodes.

The opcode map tables are organized by hex values of the upper and lower 4 bits of an opcode byte. For 1-byte encodings (Table A-2), use the four high-order bits of an opcode to index a row of the opcode table; use the four low-order bits to index a column of the table. For 2-byte opcodes beginning with 0FH (Table A-3), skip any instruction prefixes, the 0FH byte (0FH may be preceded by 66H, F2H, or F3H) and use the upper and lower 4-bit values of the next opcode byte to index table rows and columns. Similarly, for 3-byte opcodes beginning with 0F38H or 0F3AH (Table A-4), skip any instruction prefixes, 0F38H or 0F3AH and use the upper and lower 4-bit values of the third opcode byte to index table rows and columns. See Section A.2.4, "Opcode Look-up Examples for One, Two, and Three-Byte Opcodes."

When a ModR/M byte provides opcode extensions, this information qualifies opcode execution. For information on how an opcode extension in the ModR/M byte modifies the opcode map in Table A-2 and Table A-3, see Section A.4.

The escape (ESC) opcode tables for floating-point instructions identify the eight high order bits of opcodes at the top of each page. See Section A.5. If the accompanying ModR/M byte is in the range of 00H-BFH, bits 3-5 (the top row of the third table on each page) along with the reg bits of ModR/M determine the opcode. ModR/M bytes outside the range of 00H-BFH are mapped by the bottom two tables on each page of the section.

A.2 KEY TO ABBREVIATIONS

Operands are identified by a two-character code of the form Zz. The first character, an uppercase letter, specifies the addressing method; the second character, a lowercase letter, specifies the type of operand.

A.2.1 Codes for Addressing Method

The following abbreviations are used to document addressing methods:

- A Direct address: the instruction has no ModR/M byte; the address of the operand is encoded in the instruction. No base register, index register, or scaling factor can be applied (for example, far JMP (EA)).
- B The VEX.vvvv field of the VEX prefix selects a general purpose register.
- C The reg field of the ModR/M byte selects a control register (for example, MOV (0F20, 0F22)).

D	The reg field of the ModR/M byte selects a debug register (for example, MOV (0F21,0F23)).
E	A ModR/M byte follows the opcode and specifies the operand. The operand is either a general-purpose register or a memory address. If it is a memory address, the address is computed from a segment register and any of the following values: a base register, an index register, a scaling factor, a displacement.
F	EFLAGS/RFLAGS Register.
G	The reg field of the ModR/M byte selects a general register (for example, AX (000)).
H	The VEX.vvvv field of the VEX prefix selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type. For legacy SSE encodings this operand does not exist, changing the instruction to destructive form.
I	Immediate data: the operand value is encoded in subsequent bytes of the instruction.
J	The instruction contains a relative offset to be added to the instruction pointer register (for example, JMP (0E9), LOOP).
L	The upper 4 bits of the 8-bit immediate selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type. (the MSB is ignored in 32-bit mode)
M	The ModR/M byte may refer only to memory (for example, BOUND, LES, LDS, LSS, LFS, LGS, CMPX-CHG8B).
N	The R/M field of the ModR/M byte selects a packed-quadword, MMX technology register.
O	The instruction has no ModR/M byte. The offset of the operand is coded as a word or double word (depending on address size attribute) in the instruction. No base register, index register, or scaling factor can be applied (for example, MOV (A0-A3)).
P	The reg field of the ModR/M byte selects a packed quadword MMX technology register.
Q	A ModR/M byte follows the opcode and specifies the operand. The operand is either an MMX technology register or a memory address. If it is a memory address, the address is computed from a segment register and any of the following values: a base register, an index register, a scaling factor, and a displacement.
R	The R/M field of the ModR/M byte may refer only to a general register (for example, MOV (0F20-0F23)).
S	The reg field of the ModR/M byte selects a segment register (for example, MOV (8C,8E)).
U	The R/M field of the ModR/M byte selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type.
V	The reg field of the ModR/M byte selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type.
W	A ModR/M byte follows the opcode and specifies the operand. The operand is either a 128-bit XMM register, a 256-bit YMM register (determined by operand type), or a memory address. If it is a memory address, the address is computed from a segment register and any of the following values: a base register, an index register, a scaling factor, and a displacement.
X	Memory addressed by the DS:rSI register pair (for example, MOVS, CMPS, OUTS, or LODS).
Y	Memory addressed by the ES:rDI register pair (for example, MOVS, CMPS, INS, STOS, or SCAS).

A.2.2 Codes for Operand Type

The following abbreviations are used to document operand types:

- a Two one-word operands in memory or two double-word operands in memory, depending on operand-size attribute (used only by the BOUND instruction).
- b Byte, regardless of operand-size attribute.
- c Byte or word, depending on operand-size attribute.
- d Doubleword, regardless of operand-size attribute.
- dq Double-quadword, regardless of operand-size attribute.

p	32-bit, 48-bit, or 80-bit pointer, depending on operand-size attribute.
pd	128-bit or 256-bit packed double precision floating-point data.
pi	Quadword MMX technology register (for example: mm0).
ps	128-bit or 256-bit packed single precision floating-point data.
q	Quadword, regardless of operand-size attribute.
qq	Quad-Quadword (256-bits), regardless of operand-size attribute.
s	6-byte or 10-byte pseudo-descriptor.
sd	Scalar element of a 128-bit double precision floating data.
ss	Scalar element of a 128-bit single precision floating data.
si	Doubleword integer register (for example: eax).
v	Word, doubleword or quadword (in 64-bit mode), depending on operand-size attribute.
w	Word, regardless of operand-size attribute.
x	dq or qq based on the operand-size attribute.
y	Doubleword or quadword (in 64-bit mode), depending on operand-size attribute.
z	Word for 16-bit operand-size or doubleword for 32 or 64-bit operand-size.

A.2.3 Register Codes

When an opcode requires a specific register as an operand, the register is identified by name (for example, AX, CL, or ESI). The name indicates whether the register is 64, 32, 16, or 8 bits wide.

A register identifier of the form eXX or rXX is used when register width depends on the operand-size attribute. eXX is used when 16 or 32-bit sizes are possible; rXX is used when 16, 32, or 64-bit sizes are possible. For example: eAX indicates that the AX register is used when the operand-size attribute is 16 and the EAX register is used when the operand-size attribute is 32. rAX can indicate AX, EAX or RAX.

When the REX.B bit is used to modify the register specified in the reg field of the opcode, this fact is indicated by adding "/x" to the register name to indicate the additional possibility. For example, rCX/r9 is used to indicate that the register could either be rCX or r9. Note that the size of r9 in this case is determined by the operand size attribute (just as for rCX).

A.2.4 Opcode Look-up Examples for One, Two, and Three-Byte Opcodes

This section provides examples that demonstrate how opcode maps are used.

A.2.4.1 One-Byte Opcode Instructions

The opcode map for 1-byte opcodes is shown in Table A-2. The opcode map for 1-byte opcodes is arranged by row (the least-significant 4 bits of the hexadecimal value) and column (the most-significant 4 bits of the hexadecimal value). Each entry in the table lists one of the following types of opcodes:

- Instruction mnemonics and operand types using the notations listed in Section A.2
- Opcodes used as an instruction prefix

For each entry in the opcode map that corresponds to an instruction, the rules for interpreting the byte following the primary opcode fall into one of the following cases:

- A ModR/M byte is required and is interpreted according to the abbreviations listed in Section A.1 and Chapter 2, "Instruction Format," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A. Operand types are listed according to notations listed in Section A.2.
- A ModR/M byte is required and includes an opcode extension in the reg field in the ModR/M byte. Use Table A-6 when interpreting the ModR/M byte.

- Use of the ModR/M byte is reserved or undefined. This applies to entries that represent an instruction prefix or entries for instructions without operands that use ModR/M (for example: 60H, PUSH A; 06H, PUSH ES).

Example A-1. Look-up Example for 1-Byte Opcodes

Opcode 030500000000H for an ADD instruction is interpreted using the 1-byte opcode map (Table A-2) as follows:

- The first digit (0) of the opcode indicates the table row and the second digit (3) indicates the table column. This locates an opcode for ADD with two operands.
- The first operand (type Gv) indicates a general register that is a word or doubleword depending on the operand-size attribute. The second operand (type Ev) indicates a ModR/M byte follows that specifies whether the operand is a word or doubleword general-purpose register or a memory address.
- The ModR/M byte for this instruction is 05H, indicating that a 32-bit displacement follows (00000000H). The reg/opcode portion of the ModR/M byte (bits 3-5) is 000, indicating the EAX register.

The instruction for this opcode is ADD EAX, mem_op, and the offset of mem_op is 00000000H.

Some 1- and 2-byte opcodes point to group numbers (shaded entries in the opcode map table). Group numbers indicate that the instruction uses the reg/opcode bits in the ModR/M byte as an opcode extension (refer to Section A.4).

A.2.4.2 Two-Byte Opcode Instructions

The two-byte opcode map shown in Table A-3 includes primary opcodes that are either two bytes or three bytes in length. Primary opcodes that are 2 bytes in length begin with an escape opcode 0FH. The upper and lower four bits of the second opcode byte are used to index a particular row and column in Table A-3.

Two-byte opcodes that are 3 bytes in length begin with a mandatory prefix (66H, F2H, or F3H) and the escape opcode (0FH). The upper and lower four bits of the third byte are used to index a particular row and column in Table A-3 (except when the second opcode byte is the 3-byte escape opcodes 38H or 3AH; in this situation refer to Section A.2.4.3).

For each entry in the opcode map, the rules for interpreting the byte following the primary opcode fall into one of the following cases:

- A ModR/M byte is required and is interpreted according to the abbreviations listed in Section A.1 and Chapter 2, "Instruction Format," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A. The operand types are listed according to notations listed in Section A.2.
- A ModR/M byte is required and includes an opcode extension in the reg field in the ModR/M byte. Use Table A-6 when interpreting the ModR/M byte.
- Use of the ModR/M byte is reserved or undefined. This applies to entries that represent an instruction without operands that are encoded using ModR/M (for example: 0F77H, EMMS).

Example A-2. Look-up Example for 2-Byte Opcodes

Look-up opcode 0FA405000000003H for a SHLD instruction using Table A-3.

- The opcode is located in row A, column 4. The location indicates a SHLD instruction with operands Ev, Gv, and Ib. Interpret the operands as follows:
 - Ev: The ModR/M byte follows the opcode to specify a word or doubleword operand.
 - Gv: The reg field of the ModR/M byte selects a general-purpose register.
 - Ib: Immediate data is encoded in the subsequent byte of the instruction.
- The third byte is the ModR/M byte (05H). The mod and opcode/reg fields of ModR/M indicate that a 32-bit displacement is used to locate the first operand in memory and EAX as the second operand.
- The next part of the opcode is the 32-bit displacement for the destination memory operand (00000000H). The last byte stores immediate byte that provides the count of the shift (03H).
- By this breakdown, it has been shown that this opcode represents the instruction: SHLD DS:00000000H, EAX, 3.

A.2.4.3 Three-Byte Opcode Instructions

The three-byte opcode maps shown in Table A-4 and Table A-5 includes primary opcodes that are either 3 or 4 bytes in length. Primary opcodes that are 3 bytes in length begin with two escape bytes 0F38H or 0F3AH. The upper and lower four bits of the third opcode byte are used to index a particular row and column in Table A-4 or Table A-5.

Three-byte opcodes that are 4 bytes in length begin with a mandatory prefix (66H, F2H, or F3H) and two escape bytes (0F38H or 0F3AH). The upper and lower four bits of the fourth byte are used to index a particular row and column in Table A-4 or Table A-5.

For each entry in the opcode map, the rules for interpreting the byte following the primary opcode fall into the following case:

- A ModR/M byte is required and is interpreted according to the abbreviations listed in A.1 and Chapter 2, “Instruction Format,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A. The operand types are listed according to notations listed in Section A.2.

Example A-3. Look-up Example for 3-Byte Opcodes

Look-up opcode 660F3A0FC108H for a PALIGNR instruction using Table A-5.

- 66H is a prefix and 0F3AH indicate to use Table A-5. The opcode is located in row 0, column F indicating a PALIGNR instruction with operands Vdq, Wdq, and Ib. Interpret the operands as follows:
 - Vdq: The reg field of the ModR/M byte selects a 128-bit XMM register.
 - Wdq: The R/M field of the ModR/M byte selects either a 128-bit XMM register or memory location.
 - Ib: Immediate data is encoded in the subsequent byte of the instruction.
- The next byte is the ModR/M byte (C1H). The reg field indicates that the first operand is XMM0. The mod shows that the R/M field specifies a register and the R/M indicates that the second operand is XMM1.
- The last byte is the immediate byte (08H).
- By this breakdown, it has been shown that this opcode represents the instruction: PALIGNR XMM0, XMM1, 8.

A.2.4.4 VEX Prefix Instructions

Instructions that include a VEX prefix are organized relative to the 2-byte and 3-byte opcode maps, based on the VEX.mmmmm field encoding of implied 0F, 0F38H, 0F3AH, respectively. Each entry in the opcode map of a VEX-encoded instruction is based on the value of the opcode byte, similar to non-VEX-encoded instructions.

A VEX prefix includes several bit fields that encode implied 66H, F2H, F3H prefix functionality (VEX.pp) and operand size/opcode information (VEX.L). See chapter 4 for details.

Opcode tables A2-A6 include both instructions with a VEX prefix and instructions without a VEX prefix. Many entries are only made once, but represent both the VEX and non-VEX forms of the instruction. If the VEX prefix is present all the operands are valid and the mnemonic is usually prefixed with a “v”. If the VEX prefix is not present the VEX.vvvv operand is not available and the prefix “v” is dropped from the mnemonic.

A few instructions exist only in VEX form and these are marked with a superscript “v”.

Operand size of VEX prefix instructions can be determined by the operand type code. 128-bit vectors are indicated by 'dq', 256-bit vectors are indicated by 'qq', and instructions with operands supporting either 128 or 256-bit, determined by VEX.L, are indicated by 'x'. For example, the entry "VMOVUPD Vx,Wx" indicates both VEX.L=0 and VEX.L=1 are supported.

A.2.5 Superscripts Utilized in Opcode Tables

Table A-1 contains notes on particular encodings. These notes are indicated in the following opcode maps by superscripts. Gray cells indicate instruction groupings.

Table A-1. Superscripts Utilized in Opcode Tables

Superscript Symbol	Meaning of Symbol
1A	Bits 5, 4, and 3 of ModR/M byte used as an opcode extension (refer to Section A.4, "Opcode Extensions For One-Byte And Two-byte Opcodes").
1B	Use the 0F0B opcode (UD2 instruction), the 0FB9H opcode (UD1 instruction), the 0FFFH opcode (UD0 instruction), or the D6 opcode (UDB instruction) when deliberately trying to generate an invalid opcode exception (#UD).
1C	Some instructions use the same two-byte opcode. If the instruction has variations, or the opcode represents different instructions, the ModR/M byte will be used to differentiate the instruction. For the value of the ModR/M byte needed to decode the instruction, see Table A-6.
i64	The instruction is invalid or not encodable in 64-bit mode. 40 through 4F (single-byte INC and DEC) are REX prefix combinations when in 64-bit mode (use FE/FF Grp 4 and 5 for INC and DEC).
o64	Instruction is only available when in 64-bit mode.
d64	When in 64-bit mode, instruction defaults to 64-bit operand size and cannot encode 32-bit operand size.
f64	The operand size is forced to a 64-bit operand size when in 64-bit mode (prefixes that change operand size are ignored for this instruction in 64-bit mode).
v	VEX form only exists. There is no legacy SSE form of the instruction. For Integer GPR instructions it means VEX prefix required.
v1	VEX128 & SSE forms only exist (no VEX256), when can't be inferred from the data size.

A.3 ONE, TWO, AND THREE-BYTE OPCODE MAPS

See Table A-2 through Table A-5 below. The tables are multiple page presentations. Rows and columns with sequential relationships are placed on facing pages to make look-up tasks easier. Note that table footnotes are not presented on each page. Table footnotes for each table are presented on the last page of the table.

Table A-2. One-byte Opcode Map: (00H — F7H) *

	0	1	2	3	4	5	6	7
0	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, lb	rAX, lz	PUSH ES ⁶⁴	POP ES ⁶⁴
1	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, lb	rAX, lz	PUSH SS ⁶⁴	POP SS ⁶⁴
2	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, lb	rAX, lz	SEG=ES (Prefix)	DAA ⁶⁴
3	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, lb	rAX, lz	SEG=SS (Prefix)	AAA ⁶⁴
4	eAX REX	eCX REX.B	eDX REX.X	eBX REX.XB	eSP REX.R	eBP REX.RB	eSI REX.RX	eDI REX.RXB
5	rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSHA ⁶⁴ / PUSHAD ⁶⁴	POPA ⁶⁴ / POPAD ⁶⁴	BOUND ⁶⁴ Gv, Ma	ARPL ⁶⁴ Ew, Gw MOVSD ⁶⁴ Gv, Ev	SEG=FS (Prefix)	SEG=GS (Prefix)	Operand Size (Prefix)	Address Size (Prefix)
7	O	NO	B/NAE/C	NB/AE/NC	Z/E	NZ/NE	BE/NA	NBE/A
8	Eb, lb	Ev, lz	Eb, lb ⁶⁴	Ev, lb	Eb, Gb	Ev, Gv	Eb, Gb	Ev, Gv
9	NOP PAUSE(F3) XCHG r8, rAX	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
A	AL, Ob	rAX, Ov	Ob, AL	Ov, rAX	MOVS/B Yb, Xb	MOVS/W/D/Q Yv, Xv	CMPS/B Xb, Yb	CMPS/W/D/Q Xv, Yv
B	AL/R8B, lb	CL/R9B, lb	DL/R10B, lb	BL/R11B, lb	AH/R12B, lb	CH/R13B, lb	DH/R14B, lb	BH/R15B, lb
C	Eb, lb	Ev, lb	near RET ⁶⁴ lw	near RET ⁶⁴	LES ⁶⁴ Gz, Mp VEX+2byte	LDS ⁶⁴ Gz, Mp VEX+1byte	Grp 11 ^{1A} - MOV Eb, lb Ev, lz	
D	Eb, 1	Ev, 1	Eb, CL	Ev, CL	AAM ⁶⁴ lb	AAD ⁶⁴ lb	UDB ^{1B}	XLAT/ XLATB
E	LOOPNE ⁶⁴ / LOOPNZ ⁶⁴ Jb	LOOPE ⁶⁴ / LOOPZ ⁶⁴ Jb	LOOP ⁶⁴ Jb	Jrcxz ⁶⁴ / Jb	AL, lb	IN eAX, lb	OUT lb, AL	lb, eAX
F	LOCK (Prefix)	INT1	REPNE XACQUIRE (Prefix)	REP/REPE XRELEASE (Prefix)	HLT	CMC	Unary Grp 3 ^{1A} Eb Ev	

Table A-2. One-byte Opcode Map: (08H — FFH) *

	8	9	A	B	C	D	E	F
0	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH CS ⁱ⁶⁴	2-byte escape (Table A-3)
1	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH DS ⁱ⁶⁴	POP DS ⁱ⁶⁴
2	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	SEG=CS (Prefix)	DAS ⁱ⁶⁴
3	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	SEG=DS (Prefix)	AAS ⁱ⁶⁴
4	eAX REX.W	eCX REX.WB	eDX REX.WX	eBX REX.WXB	eSP REX.WR	eBP REX.WRB	eSI REX.WRX	eDI REX.WRXB
5	rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSH ^{d64} Iz	IMUL Gv, Ev, Iz	PUSH ^{d64} Ib	IMUL Gv, Ev, Ib	INS/ INSB Yb, DX	INS/ INSW/ INSD Yz, DX	OUTS/ OUTSB DX, Xb	OUTS/ OUTSW/ OUTSD DX, Xz
7	S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
8	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	MOV Ev, Sw	LEA Gv, M	MOV Sw, Ew	Grp 1A ^{1A} POP ^{d64} Ev
9	CBW/ CWDE/ CDQE	CWD/ CDQ/ CQO	far CALL ⁱ⁶⁴ Ap	FWAIT/ WAIT	PUSHF/D/Q ^{d64} / Fv	POPF/D/Q ^{d64} / Fv	SAHF	LAHF
A	TEST AL, Ib	rAX, Iz	STOS/B Yb, AL	STOS/W/D/Q Yv, rAX	LODS/B AL, Xb	LODS/W/D/Q rAX, Xv	SCAS/B AL, Yb	SCAS/W/D/Q rAX, Yv
B	rAX/r8, Iv	rCX/r9, Iv	rDX/r10, Iv	rBX/r11, Iv	rSP/r12, Iv	rBP/r13, Iv	rSI/r14, Iv	rDI/r15, Iv
C	ENTER Iw, Ib	LEAVE ^{d64}	far RET Iw	far RET	INT3	INT Ib	INTO ⁱ⁶⁴	IRET/D/Q
D	ESC (Escape to coprocessor instruction set)							
E	near CALL ^{f64} Jz	near ^{f64} Jz	JMP far ⁱ⁶⁴ Ap	short ^{f64} Jb	AL, DX	eAX, DX	DX, AL	DX, eAX
F	CLC	STC	CLI	STI	CLD	STD	INC/DEC Grp 4 ^{1A}	INC/DEC Grp 5 ^{1A}

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-3. Two-byte Opcode Map: 00H – 77H (First Byte is 0FH) *

	px	0	1	2	3	4	5	6	7
0		Grp 6 ^{1A}	Grp 7 ^{1A}	LAR Gv, Ew	LSL Gv, Ew		SYSCALL ^{o64}	CLTS	SYSRET ^{o64}
1		vmovups Vps, Wps	vmovups Wps, Vps	vmovlps Vq, Hq, Mq vmovhlps Vq, Hq, Uq	vmovlps Mq, Vq	vunpcklps Vx, Hx, Wx	vunpckhps Vx, Hx, Wx	vmovhps ^{v1} Vdq, Hq, Mq vmovhlps Vdq, Hq, Uq	vmovhps ^{v1} Mq, Vq
	66	vmovupd Vpd, Wpd	vmovupd Wpd, Vpd	vmovlpd Vq, Hq, Mq	vmovlpd Mq, Vq	vunpcklpd Vx, Hx, Wx	vunpckhpd Vx, Hx, Wx	vmovhpd ^{v1} Vdq, Hq, Mq	vmovhpd ^{v1} Mq, Vq
	F3	vmovss Vx, Hx, Wss	vmovss Wss, Hx, Vss	vmovsldup Vx, Wx				vmovshdup Vx, Wx	
	F2	vmovsd Vx, Hx, Wsd	vmovsd Wsd, Hx, Vsd	vmovddup Vx, Wx					
2		MOV Rd, Cd	MOV Rd, Dd	MOV Cd, Rd	MOV Dd, Rd				
3		WRMSR	RDTSC	RDMSR	RDPNC	SYSENTER	SYSEXIT		GETSEC
4		CMOVcc, (Gv, Ev) - Conditional Move							
		O	NO	B/C/NAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
5		vmovmskps Gy, Ups	vsqrtps Vps, Wps	vrsqrtps Vps, Wps	vrcpps Vps, Wps	vandps Vps, Hps, Wps	vandnps Vps, Hps, Wps	vorps Vps, Hps, Wps	vxorps Vps, Hps, Wps
	66	vmovmskpd Gy, Upd	vsqrtpd Vpd, Wpd			vandpd Vpd, Hpd, Wpd	vandnpd Vpd, Hpd, Wpd	vorpd Vpd, Hpd, Wpd	vxorpd Vpd, Hpd, Wpd
	F3		vsqrtss Vss, Hss, Wss	vrsqrtss Vss, Hss, Wss	vrcpss Vss, Hss, Wss				
	F2		vsqrtsd Vsd, Hsd, Wsd						
6		punpcklbw Pq, Qd	punpcklwd Pq, Qd	punpckldq Pq, Qd	packsswb Pq, Qq	pcmpgtb Pq, Qq	pcmpgtw Pq, Qq	pcmpgtd Pq, Qq	packuswb Pq, Qq
	66	vpunpcklbw Vx, Hx, Wx	vpunpcklwd Vx, Hx, Wx	vpunpckldq Vx, Hx, Wx	vpacksswb Vx, Hx, Wx	vpcmpgtb Vx, Hx, Wx	vpcmpgtw Vx, Hx, Wx	vpcmpgtd Vx, Hx, Wx	vpackuswb Vx, Hx, Wx
	F3								
7		pshufw Pq, Qq, Ib	(Grp 12 ^{1A})	(Grp 13 ^{1A})	(Grp 14 ^{1A})	pcmpeqb Pq, Qq	pcmpeqw Pq, Qq	pcmpeqd Pq, Qq	emms vzeroupper ^v vzeroall ^v
	66	vpshufd Vx, Wx, Ib				vpcmpeqb Vx, Hx, Wx	vpcmpeqw Vx, Hx, Wx	vpcmpeqd Vx, Hx, Wx	
	F3	vpshufhw Vx, Wx, Ib							
	F2	vpshufw Vx, Wx, Ib							

Table A-3. Two-byte Opcode Map: 08H – 7FH (First Byte is 0FH) *

	px	8	9	A	B	C	D	E	F
0		INVD	WBINVD		2-byte Illegal Opcodes UD2 ^{1B}		prefetchw(/1) Ev		
	F3		WBNOINVD						
1		Prefetch ^{1C} (Grp 16 ^{1A})	Reserved-NOP	bndldx	bndstx	(Grp 18 ^{1A})	Reserved-NOP		NOP /0 Ev
	66			bndmov	bndmov				
	F3			bndcl	bndmk				
	F2			bndcu	bndcn				

Table A-3. Two-byte Opcode Map: 08H – 7FH (First Byte is 0FH) *

	pxf	8	9	A	B	C	D	E	F
2		vmovaps Vps, Wps	vmovaps Wps, Vps	cvtpi2ps Vps, Qpi	vmovntps Mps, Vps	cvtps2pi Ppi, Wps	cvtps2pi Ppi, Wps	vucmiss Vss, Wss	vcomiss Vss, Wss
	66	vmovapd Vpd, Wpd	vmovapd Wpd, Vpd	cvtpi2pd Vpd, Qpi	vmovntpd Mpd, Vpd	cvtpd2pi Ppi, Wpd	cvtpd2pi Qpi, Wpd	vucmisd Vsd, Wsd	vcomisd Vsd, Wsd
	F3			vcvtss2ss Vss, Hss, Ey		vcvtss2si Gy, Wss	vcvtss2si Gy, Wss		
	F2			vcvtss2sd Vsd, Hsd, Ey		vcvtss2si Gy, Wsd	vcvtss2si Gy, Wsd		
3		3-byte escape (Table A-4)		3-byte escape (Table A-5)					
4		CMOVcc(Gv, Ev) - Conditional Move							
		S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
5		vaddps Vps, Hps, Wps	vmulps Vps, Hps, Wps	vcvtps2pd Vpd, Wps	vcvtdq2ps Vps, Wdq	vsubps Vps, Hps, Wps	vminps Vps, Hps, Wps	vdivps Vps, Hps, Wps	vmaxps Vps, Hps, Wps
	66	vaddpd Vpd, Hpd, Wpd	vmulpd Vpd, Hpd, Wpd	vcvtpd2ps Vps, Wpd	vcvtps2dq Vdq, Wps	vsubpd Vpd, Hpd, Wpd	vminpd Vpd, Hpd, Wpd	vdivpd Vpd, Hpd, Wpd	vmaxpd Vpd, Hpd, Wpd
	F3	vaddss Vss, Hss, Wss	vmulss Vss, Hss, Wss	vcvtss2sd Vsd, Hx, Wss	vcvtps2dq Vdq, Wps	vsubss Vss, Hss, Wss	vminss Vss, Hss, Wss	vdivss Vss, Hss, Wss	vmaxss Vss, Hss, Wss
	F2	vaddsd Vsd, Hsd, Wsd	vmulsd Vsd, Hsd, Wsd	vcvtss2ss Vss, Hx, Wsd		vsubsd Vsd, Hsd, Wsd	vminsd Vsd, Hsd, Wsd	vdivsd Vsd, Hsd, Wsd	vmaxsd Vsd, Hsd, Wsd
6		punpckhbw Pq, Qd	punpckhwd Pq, Qd	punpckhdq Pq, Qd	packssdw Pq, Qd			movd/q Pd, Ey	movq Pq, Qq
	66	vpunpckhbw Vx, Hx, Wx	vpunpckhwd Vx, Hx, Wx	vpunpckhdq Vx, Hx, Wx	vpackssdw Vx, Hx, Wx	vpunpckldq Vx, Hx, Wx	vpunpckhdq Vx, Hx, Wx	vmovd/q Vy, Ey	vmovdqa Vx, Wx
	F3								vmovdqu Vx, Wx
7		VMREAD Ey, Gy	VMWRITE Gy, Ey					movd/q Ey, Pd	movq Qq, Pq
	66					vhaddpd Vpd, Hpd, Wpd	vhsubpd Vpd, Hpd, Wpd	vmovd/q Ey, Vy	vmovdqa Wx, Vx
	F3							vmovq Vq, Wq	vmovdqu Wx, Vx
	F2					vhaddps Vps, Hps, Wps	vhsubps Vps, Hps, Wps		

Table A-3. Two-byte Opcode Map: 80H — F7H (First Byte is 0FH) *

	pxf	0	1	2	3	4	5	6	7
8		Jcc ^{f64} , Jz - Long-displacement jump on condition							
		O	NO	B/CNAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
9		SETcc, Eb - Byte Set on condition							
		O	NO	B/CNAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
A		PUSH ^{d64} FS	POP ^{d64} FS	CPUID	BT Ev, Gv	SHLD Ev, Gv, Ib	SHLD Ev, Gv, CL		
B		CMPXCHG Eb, Gb		LSS Gv, Mp	BTR Ev, Gv	LFS Gv, Mp	LGS Gv, Mp	MOVZX Gv, Eb Gv, Ew	
C		XADD Eb, Gb	XADD Ev, Gv	vcmpps Vps, Hps, Wps, Ib	movnti My, Gy	pinsrw Pq, Ry/Mw, Ib	pextrw Gd, Nq, Ib	vshufps Vps, Hps, Wps, Ib	Grp 9 ^{1A}
	66			vcmpdp Vpd, Hpd, Wpd, Ib		vpinsrw Vdq, Hdq, Ry/Mw, Ib	vpextrw Gd, Udq, Ib	vshufpd Vpd, Hpd, Wpd, Ib	
	F3			vcmpss Vss, Hss, Wss, Ib					
	F2			vcmpsdp Vsd, Hsd, Wsd, Ib					
D			psrlw Pq, Qq	psrld Pq, Qq	psrlq Pq, Qq	paddq Pq, Qq	pmullw Pq, Qq		pmovmskb Gd, Nq
	66	vaddsubpd Vpd, Hpd, Wpd	vpsrlw Vx, Hx, Wx	vpsrld Vx, Hx, Wx	vpsrlq Vx, Hx, Wx	vpaddq Vx, Hx, Wx	vpmullw Vx, Hx, Wx	vmovq Wq, Vq	vpmovmskb Gd, Ux
	F3							movq2dq Vdq, Nq	
	F2	vaddsubps Vps, Hps, Wps						movdq2q Pq, Uq	
E		pavgb Pq, Qq	psraw Pq, Qq	psrad Pq, Qq	pavgw Pq, Qq	pmulhuw Pq, Qq	pmulhw Pq, Qq		movntq Mq, Pq
	66	vpavgb Vx, Hx, Wx	vpsraw Vx, Hx, Wx	vpsrad Vx, Hx, Wx	vpavgw Vx, Hx, Wx	vpmulhuw Vx, Hx, Wx	vpmulhw Vx, Hx, Wx	vcvttd2dq Vx, Wpd	vmovntdq Mx, Vx
	F3							vcvtdq2pd Vx, Wpd	
	F2							vcvtpd2dq Vx, Wpd	
F			psllw Pq, Qq	pslld Pq, Qq	psllq Pq, Qq	pmuludq Pq, Qq	pmaddwd Pq, Qq	psadbw Pq, Qq	maskmovq Pq, Nq
	66		vpsllw Vx, Hx, Wx	vpslld Vx, Hx, Wx	vpsllq Vx, Hx, Wx	vpmuludq Vx, Hx, Wx	vpmaddwd Vx, Hx, Wx	vpsadbw Vx, Hx, Wx	vmaskmovdqu Vdq, Udq
	F2	vlddqu Vx, Mx							

Table A-3. Two-byte Opcode Map: 88H — FFH (First Byte is 0FH) *

	pxf	8	9	A	B	C	D	E	F
8		Jcc ⁶⁴ , Jz - Long-displacement jump on condition							
		S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
9		SETcc, Eb - Byte Set on condition							
		S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
A		PUSH ^{d64} GS	POP ^{d64} GS	RSM	BTS Ev, Gv	SHRD Ev, Gv, Ib	SHRD Ev, Gv, CL	(Grp 15 ^{1A}) ^{1C}	IMUL Gv, Ev
B		JMPE (reserved for emulator on IPF)	Grp 10 ^{1A} Invalid Opcode ^{1B}	Grp 8 ^{1A} Ev, Ib	BTC Ev, Gv	BSF Gv, Ev	BSR Gv, Ev	MOVsx Gv, Eb Gv, Ew	
	F3	POPCNT Gv, Ev				TZCNT Gv, Ev	LZCNT Gv, Ev		
C		BSWAP							
		RAX/EAX/ R8/R8D	RCX/ECX/ R9/R9D	RDX/EDX/ R10/R10D	RBX/EBX/ R11/R11D	RSP/ESP/ R12/R12D	RBP/EBP/ R13/R13D	RSI/ESI/ R14/R14D	RDI/EDI/ R15/R15D
D		psubusb Pq, Qq	psubusw Pq, Qq	pminub Pq, Qq	pand Pq, Qq	paddusb Pq, Qq	paddusw Pq, Qq	pmaxub Pq, Qq	pandn Pq, Qq
	66	vpsubusb Vx, Hx, Wx	vpsubusw Vx, Hx, Wx	vpminub Vx, Hx, Wx	vpand Vx, Hx, Wx	vpaddusb Vx, Hx, Wx	vpaddusw Vx, Hx, Wx	vpmaxub Vx, Hx, Wx	vpandn Vx, Hx, Wx
	F3								
	F2								
E		psubsb Pq, Qq	psubsw Pq, Qq	pminsw Pq, Qq	por Pq, Qq	paddsb Pq, Qq	paddsw Pq, Qq	pmaxsw Pq, Qq	pxor Pq, Qq
	66	vpsubsb Vx, Hx, Wx	vpsubsw Vx, Hx, Wx	vpminsw Vx, Hx, Wx	vpior Vx, Hx, Wx	vpaddsb Vx, Hx, Wx	vpaddsw Vx, Hx, Wx	vpmaxsw Vx, Hx, Wx	vpxor Vx, Hx, Wx
	F3								
	F2								
F		psubb Pq, Qq	psubw Pq, Qq	psubd Pq, Qq	psubq Pq, Qq	paddb Pq, Qq	paddw Pq, Qq	paddd Pq, Qq	UD0 ^{1B}
	66	vpsubb Vx, Hx, Wx	vpsubw Vx, Hx, Wx	vpsubd Vx, Hx, Wx	vpsubq Vx, Hx, Wx	vpaddb Vx, Hx, Wx	vpaddw Vx, Hx, Wx	vpaddd Vx, Hx, Wx	
	F2								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-4. Three-byte Opcode Map: 00H — F7H (First Two Bytes are 0F 38H) *

	pxf	0	1	2	3	4	5	6	7
0		pshufb Pq, Qq	phaddw Pq, Qq	phadd Pq, Qq	phaddsw Pq, Qq	pmaddubsw Pq, Qq	phsubw Pq, Qq	phsubd Pq, Qq	phsubsw Pq, Qq
	66	vpshufb Vx, Hx, Wx	vphaddw Vx, Hx, Wx	vphadd Vx, Hx, Wx	vphaddsw Vx, Hx, Wx	vpmaddubsw Vx, Hx, Wx	vphsubw Vx, Hx, Wx	vphsubd Vx, Hx, Wx	vphsubsw Vx, Hx, Wx
1		pblendvb Vdq, Wdq			vcvtph2ps ^V Vx, Wx, lb	blendvps Vdq, Wdq	blendvpd Vdq, Wdq	vpermps ^V Vqq, Hqq, Wqq	vptest Vx, Wx
	66								
2	66	vpmovsxbw Vx, Ux/Mq	vpmovsxbd Vx, Ux/Md	vpmovsxbq Vx, Ux/Mw	vpmovsxd Vx, Ux/Mq	vpmovsxwq Vx, Ux/Md	vpmovsxdq Vx, Ux/Mq		
3	66	vpmovzxbw Vx, Ux/Mq	vpmovzxbd Vx, Ux/Md	vpmovzxbq Vx, Ux/Mw	vpmovzxd Vx, Ux/Mq	vpmovzxwq Vx, Ux/Md	vpmovzxdq Vx, Ux/Mq	vpermd ^V Vqq, Hqq, Wqq	vpcmpgtq Vx, Hx, Wx
4	66	vpmulld Vx, Hx, Wx	vphminposuw Vdq, Wdq				vpsrlvd/q ^V Vx, Hx, Wx	vpsravd ^V Vx, Hx, Wx	vpslvd/q ^V Vx, Hx, Wx
5									
6									
7									
8	66	INVEPT Gy, Mdq	INVVPID Gy, Mdq	INVPCID Gy, Mdq					
9	66	vgatherdd/q ^V Vx,Hx,Wx	vgatherqd/q ^V Vx,Hx,Wx	vgatherdps/d ^V Vx,Hx,Wx	vgatherqps/d ^V Vx,Hx,Wx			vfmaddsub132ps/d ^V Vx,Hx,Wx	vfmsubadd132ps/d ^V Vx,Hx,Wx
A	66							vfmaddsub213ps/d ^V Vx,Hx,Wx	vfmsubadd213ps/d ^V Vx,Hx,Wx
B	66							vfmaddsub231ps/d ^V Vx,Hx,Wx	vfmsubadd231ps/d ^V Vx,Hx,Wx
C									
D									
E									
F		MOVBE Gy, My	MOVBE My, Gy	ANDN ^V Gy, By, Ey	Grp 17 ^{1A}		BZHI ^V Gy, Ey, By		BEXTR ^V Gy, Ey, By
	66	MOVBE Gw, Mw	MOVBE Mw, Gw					ADCX Gy, Ey	SHLX ^V Gy, Ey, By
	F3						PEXT ^V Gy, By, Ey	ADOX Gy, Ey	SARX ^V Gy, Ey, By
	F2	CRC32 Gd, Eb	CRC32 Gd, Ey				PDEP ^V Gy, By, Ey	MULX ^V By,Gy,rDX,Ey	SHRX ^V Gy, Ey, By
	66 & F2	CRC32 Gd, Eb	CRC32 Gd, Ew						

Table A-4. Three-byte Opcode Map: 08H — FFH (First Two Bytes are 0F 38H) *

	pxf	8	9	A	B	C	D	E	F
0		psignb Pq, Qq	psignw Pq, Qq	psignd Pq, Qq	pmulhrsw Pq, Qq				
	66	vpsignb Vx, Hx, Wx	vpsignw Vx, Hx, Wx	vpsignd Vx, Hx, Wx	vpmulhrsw Vx, Hx, Wx	vpermilps ^v Vx,Hx,Wx	vpermilpd ^v Vx,Hx,Wx	vtestps ^v Vx, Wx	vtestpd ^v Vx, Wx
1						pabsb Pq, Qq	pabsw Pq, Qq	pabsd Pq, Qq	
	66	vbroadcastss ^v Vx, Wd	vbroadcastsd ^v Vqq, Wq	vbroadcastf128 ^v Vqq, Mdq		vpabsb Vx, Wx	vpabsw Vx, Wx	vpabsd Vx, Wx	
2	66	vpmuldq Vx, Hx, Wx	vpcmpqq Vx, Hx, Wx	vmovntdqa Vx, Mx	vpackusdw Vx, Hx, Wx	vmaskmovps ^v Vx,Hx,Mx	vmaskmovpd ^v Vx,Hx,Mx	vmaskmovps ^v Mx,Hx,Vx	vmaskmovpd ^v Mx,Hx,Vx
3	66	vpminsb Vx, Hx, Wx	vpmins d Vx, Hx, Wx	vpminuw Vx, Hx, Wx	vpminud Vx, Hx, Wx	vpmaxsb Vx, Hx, Wx	vpmaxsd Vx, Hx, Wx	vpmaxuw Vx, Hx, Wx	vpmaxud Vx, Hx, Wx
4									
5	66	vpbroadcast ^v Vx, Wx	vpbroadcastq ^v Vx, Wx	vpbroadcasti128 ^v Vqq, Mdq					
6									
7	66	vpbroadcastb ^v Vx, Wx	vpbroadcastw ^v Vx, Wx						
8	66					vpmaskmovd/q ^v Vx,Hx,Mx		vpmaskmovd/q ^v Mx,Vx,Hx	
9	66	vfmadd132ps/d ^v Vx, Hx, Wx	vfmadd132ss/d ^v Vx, Hx, Wx	vfmsub132ps/d ^v Vx, Hx, Wx	vfmsub132ss/d ^v Vx, Hx, Wx	vfnmadd132ps/d ^v Vx, Hx, Wx	vfnmadd132ss/d ^v Vx, Hx, Wx	vfmsub132ps/d ^v Vx, Hx, Wx	vfmsub132ss/d ^v Vx, Hx, Wx
A	66	vfmadd213ps/d ^v Vx, Hx, Wx	vfmadd213ss/d ^v Vx, Hx, Wx	vfmsub213ps/d ^v Vx, Hx, Wx	vfmsub213ss/d ^v Vx, Hx, Wx	vfnmadd213ps/d ^v Vx, Hx, Wx	vfnmadd213ss/d ^v Vx, Hx, Wx	vfmsub213ps/d ^v Vx, Hx, Wx	vfmsub213ss/d ^v Vx, Hx, Wx
B	66	vfmadd231ps/d ^v Vx, Hx, Wx	vfmadd231ss/d ^v Vx, Hx, Wx	vfmsub231ps/d ^v Vx, Hx, Wx	vfmsub231ss/d ^v Vx, Hx, Wx	vfnmadd231ps/d ^v Vx, Hx, Wx	vfnmadd231ss/d ^v Vx, Hx, Wx	vfmsub231ps/d ^v Vx, Hx, Wx	vfmsub231ss/d ^v Vx, Hx, Wx
C		sha1nexte Vdq,Wdq	sha1msg1 Vdq,Wdq	sha1msg2 Vdq,Wdq	sha256rds2 Vdq,Wdq	sha256msg1 Vdq,Wdq	sha256msg2 Vdq,Wdq		
	66								
D	66				VAESIMC Vdq, Wdq	VAESEC Vdq,Hdq,Wdq	VAESENCLAST Vdq,Hdq,Wdq	VAESDEC Vdq,Hdq,Wdq	VAESDECLAST Vdq,Hdq,Wdq
E									
F									
	66								
	F3								
	F2								
	66 & F2								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-5. Three-byte Opcode Map: 00H — F7H (First two bytes are 0F 3AH) *

	px	0	1	2	3	4	5	6	7
0	66	vpermq ^v Vqq, Wqq, lb	vpermpd ^v Vqq, Wqq, lb	vpblendd ^v Vx, Hx, Wx, lb		vpermilps ^v Vx, Wx, lb	vpermilpd ^v Vx, Wx, lb	vperm2f128 ^v Vqq, Hqq, Wqq, lb	
1	66					vpextrb Rd/Mb, Vdq, lb	vpextrw Rd/Mw, Vdq, lb	vpextrd/q Ey, Vdq, lb	vextractps Ed, Vdq, lb
2	66	vpinsrb Vdq, Hdq, Ry/Mb, lb	vinsertps Vdq, Hdq, Udq/Md, lb	vpinsrd/q Vdq, Hdq, Ey, lb					
3									
4	66	vdpps Vx, Hx, Wx, lb	vdppd Vdq, Hdq, Wdq, lb	vmepsadbw Vx, Hx, Wx, lb		vpcimulqdd Vdq, Hdq, Wdq, lb		vperm2i128 ^v Vqq, Hqq, Wqq, lb	
5									
6	66	vpcmpestrm Vdq, Wdq, lb	vpcmpestri Vdq, Wdq, lb	vpcmpistrm Vdq, Wdq, lb	vpcmpistri Vdq, Wdq, lb				
7									
8									
9									
A									
B									
C									
D									
E									
F	F2	RORX ^v Gy, Ey, lb							

Table A-5. Three-byte Opcode Map: 08H — FFH (First Two Bytes are 0F 3AH) *

	px	8	9	A	B	C	D	E	F
0									palignr Pq, Qq, Ib
	66	vroundps Vx, Wx, Ib	vroundpd Vx, Wx, Ib	vroundss Vss, Wss, Ib	vroundsd Vsd, Wsd, Ib	vblendps Vx, Hx, Wx, Ib	vblendpd Vx, Hx, Wx, Ib	vpblendw Vx, Hx, Wx, Ib	vpalignr Vx, Hx, Wx, Ib
1	66	vinserftf128 ^v Vqq, Hqq, Wqq, Ib	vextractf128 ^v Wdq, Vqq, Ib				vcvtps2ph ^v Wx, Vx, Ib		
2									
3	66	vinserftf128 ^v Vqq, Hqq, Wqq, Ib	vextractf128 ^v Wdq, Vqq, Ib						
4	66			vblendvps ^v Vx, Hx, Wx, Lx	vblendvpd ^v Vx, Hx, Wx, Lx	vpblendvb ^v Vx, Hx, Wx, Lx			
5									
6									
7									
8									
9									
A									
B									
C						sha1m4s4 Vdq, Wdq, Ib			
D	66								VAESKEYGEN Vdq, Wdq, Ib
E									
F									

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.4 OPCODE EXTENSIONS FOR ONE-BYTE AND TWO-BYTE OPCODES

Some 1-byte and 2-byte opcodes use bits 3-5 of the ModR/M byte (the nnn field in Figure A-1) as an extension of the opcode.

mod	nnn	R/M

Figure A-1. ModR/M Byte nnn Field (Bits 5, 4, and 3)

Opcodes that have opcode extensions are indicated in Table A-6 and organized by group number. Group numbers (from 1 to 16, second column) provide a table entry point. The encoding for the r/m field for each instruction can be established using the third column of the table.

A.4.1 Opcode Look-up Examples Using Opcode Extensions

An Example is provided below.

Example A-4. Interpreting an ADD Instruction

An ADD instruction with a 1-byte opcode of 80H is a Group 1 instruction:

- Table A-6 indicates that the opcode extension field encoded in the ModR/M byte for this instruction is 000B.
- The r/m field can be encoded to access a register (11B) or a memory address using a specified addressing mode (for example: mem = 00B, 01B, 10B).

Example A-5. Looking Up 0F01C3H

Look up opcode 0F01C3 for a VMRESUME instruction by using Table A-2, Table A-3, and Table A-6:

- 0F indicates that this instruction is in the 2-byte opcode map.
- 01 (row 0, column 1 in Table A-3) reveals that this opcode is in Group 7 of Table A-6.
- C3 is the ModR/M byte. The first two bits of C3 are 11B. This tells us to look at the second of the Group 7 rows in Table A-6.
- The Op/Reg bits [5,4,3] are 000B. This tells us to look in the 000 column for Group 7.
- Finally, the R/M bits [2,1,0] are 011B. This identifies the opcode as the VMRESUME instruction.

A.4.2 Opcode Extension Tables

See Table A-6 below.

Table A-6. Opcode Extensions for One- and Two-byte Opcodes by Group Number *

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
80-83	1	mem, 11B		ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
8F	1A	mem, 11B		POP							
C0,C1 reg, imm D0, D1 reg, 1 D2, D3 reg, CL	2	mem, 11B		ROL	ROR	RCL	RCR	SHL/SAL	SHR		SAR
F6, F7	3	mem, 11B		TEST lb/lz		NOT	NEG	MUL AL/rAX	IMUL AL/rAX	DIV AL/rAX	IDIV AL/rAX
FE	4	mem, 11B		INC Eb	DEC Eb						
FF	5	mem, 11B		INC Ev	DEC Ev	near CALL ^{f64} Ev	far CALL Ep	near JMP ^{f64} Ev	far JMP Mp	PUSH ^{d64} Ev	
0F 00	6	mem, 11B		SLDT Rv/Mw	STR Rv/Mw	LLDT Ew	LTR Ew	VERR Ew	VERW Ew		
			F2							LKGS ^{o64} Ew	
0F 01	7	mem		SGDT Ms	SIDT Ms	LGDT Ms	LIDT Ms	SMSW Mw/Rv		LMSW Ew	INVLPB Mb
			F3						RSTORSSP Mq		
		11B		VMCALL (001) VMLAUNCH (010) VMRESUME (011) VMXOFF (100) PCONFIG (101) WRMSRNS (110) PBNDKB (111)	MONITOR (000) MWAIT (001) CLAC (010) STAC (011) ENCLS (111)	XGETBV (000) XSETBV (001) VMFUNC (100) XEND (101) XTEST (110) ENCLU (111)			SERIALIZE (000) RDPKRU (110) WRPKRU (111)		SWAPGS ^{o64} (000) RDTSCP (001)
			66		TDCALL (100) SEAMRET (101) SEAMOPS (110) SEAMCALL ^{o64} (111)						
			F2	RDMSRLIST (110) ^{o64}	ERETS ^{o64} (010)				XSUSLDRK (000) XRESLDRK (001)		
			F3	WRMSRLIST (110) ^{o64}	ERETU ^{o64} (010)				SETSSBSY (000) SAVEPREVSSP (010) UIRET (100) CLUI (110) STUI (111)		
0F BA	8	mem, 11B						BT	BTS	BTR	BTC
0F C7	9	mem			CMPXCH8B Mq CMPXCHG16B Mdq		XRSTORS	XSAVEC	XSAVES	VMPTRLD Mq	VMPTRST Mq
			66							VMCLEAR Mq	
			F3							VMXON Mq	
		11B	F3							RDRAND Rv RDSEED Rv	
0F B9	10	mem		UD1 ^{1B}							
C6	11	mem		MOV Eb, lb							
		11B									XABORT (000) lb
C7		mem		MOV Ev, lz							
		11B									XBEGIN (000) Jz

Table A-6. Opcode Extensions for One- and Two-byte Opcodes by Group Number * (Contd.)

0F 71	12	mem									
		11B				psrlw Nq, lb		psraw Nq, lb		psllw Nq, lb	
			66			vpsrlw Hx,Ux,lb		vpsraw Hx,Ux,lb		vpsllw Hx,Ux,lb	
0F 72	13	mem									
		11B				psrld Nq, lb		psrad Nq, lb		pslld Nq, lb	
			66			vpsrld Hx,Ux,lb		vpsrad Hx,Ux,lb		vpslld Hx,Ux,lb	
0F 73	14	mem									
		11B				psrlq Nq, lb				psllq Nq, lb	
			66			vpsrlq Hx,Ux,lb	vpsrldq Hx,Ux,lb			vpsllq Hx,Ux,lb	vpslldq Hx,Ux,lb

Table A-6. Opcode Extensions for One- and Two-byte Opcodes by Group Number * (Contd.)

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
0F AE	15	mem		FXSAVE	FXRSTOR	LDMXCSR	STMXCSR	XSAVE	XRSTOR	XSAVEOPT	CLFLUSH
			66						CLWB	CLFLUSHOPT	
			F3				PTWRITE My		CLRSSBSY		
		11B						LFENCE (000)	MFENCE (000)	SFENCE (000)	
			66						TPAUSE Rd		
			F2						UMWAIT Rd		
	F3	RDFSBASE Ry	RDGSBASE Ry	WRFSBASE Ry	WRGSBASE Ry	PTWRITE Ry	INCSSP	UMONITOR Rv			
0F 18	16			prefetch NTA	prefetch T0	prefetch T1	prefetch T2	Reserved NOP			
		11B		Reserved NOP							
VEX.0F38 F3	17	mem			BLSR ^v By, Ey	BLSMSK ^v By, Ey	BLSI ^v By, Ey				
		11B									
0F 1C	18	mem		CLDEMOT	Reserved NOP						
			66, F2, F3	Reserved NOP							
		11B	any	Reserved NOP							

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5 ESCAPE OPCODE INSTRUCTIONS

Opcode maps for coprocessor escape instruction opcodes (x87 floating-point instruction opcodes) are in Table A-7 through Table A-22. These maps are grouped by the first byte of the opcode, from D8-DF. Each of these opcodes has a ModR/M byte. If the ModR/M byte is within the range of 00H-BFH, bits 3-5 of the ModR/M byte are used as an opcode extension, similar to the technique used for 1- and 2-byte opcodes (see A.4). If the ModR/M byte is outside the range of 00H through BFH, the entire ModR/M byte is used as an opcode extension.

A.5.1 Opcode Look-up Examples for Escape Instruction Opcodes

Examples are provided below.

Example A-6. Opcode with ModR/M Byte in the 00H through BFH Range

DD0504000000H can be interpreted as follows:

- The instruction encoded with this opcode can be located in Section . Since the ModR/M byte (05H) is within the 00H through BFH range, bits 3 through 5 (000) of this byte indicate the opcode for an FLD double-real instruction (see Table A-9).
- The double-real value to be loaded is at 00000004H (the 32-bit displacement that follows and belongs to this opcode).

Example A-7. Opcode with ModR/M Byte outside the 00H through BFH Range

D8C1H can be interpreted as follows:

- This example illustrates an opcode with a ModR/M byte outside the range of 00H through BFH. The instruction can be located in Section A.4.
- In Table A-8, the ModR/M byte C1H indicates row C, column 1 (the FADD instruction using ST(0), ST(1) as operands).

A.5.2 Escape Opcode Instruction Tables

Tables are listed below.

A.5.2.1 Escape Opcodes with D8 as First Byte

Table A-7 and A-8 contain maps for the escape instruction opcodes that begin with D8H. Table A-7 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-7. D8 Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte (refer to Figure A.4)							
000B	001B	010B	011B	100B	101B	110B	111B
FADD single-real	FMUL single-real	FCOM single-real	FCOMP single-real	FSUB single-real	FSUBR single-real	FDIV single-real	FDIVR single-real

NOTES:

- * All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-8 shows the map if the ModR/M byte is outside the range of 00H-BFH. Here, the first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-8. D8 Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FADD							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCOM							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),T(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E	FSUB							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F	FDIV							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

	8	9	A	B	C	D	E	F
C	FMUL							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCOMP							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),T(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E	FSUBR							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F	FDIVR							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.2 Escape Opcodes with D9 as First Byte

Table A-9 and A-10 contain maps for escape instruction opcodes that begin with D9H. Table A-9 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-9. D9 Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FLD single-real		FST single-real	FSTP single-real	FLDENV 14/28 bytes	FLDCW 2 bytes	FSTENV 14/28 bytes	FSTCW 2 bytes

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-10 shows the map if the ModR/M byte is outside the range of 00H-BFH. Here, the first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-10. D9 Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FLD							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FNOP							
E	FCHS	FABS			FTST	FXAM		
F	F2XM1	FYL2X	FPTAN	FPATAN	EXTRACT	FPREM1	FDECSTP	FINCSTP

	8	9	A	B	C	D	E	F
C	FXCH							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D								
E	FLD1	FLDL2T	FLDL2E	FLDPI	FLDLG2	FLDLN2	FLDZ	
F	FPREM	FYL2XP1	FSQRT	FSINCOS	FRNDINT	FSCALE	FSIN	FCOS

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.3 Escape Opcodes with DA as First Byte

Table A-11 and A-12 contain maps for escape instruction opcodes that begin with DAH. Table A-11 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-11. DA Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FIADD dword-integer	FIMUL dword-integer	FICOM dword-integer	FICOMP dword-integer	FISUB dword-integer	FISUBR dword-integer	FIDIV dword-integer	FIDIVR dword-integer

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-12 shows the map if the ModR/M byte is outside the range of 00H-BFH. Here, the first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-12. DA Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FCMOVB							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVBE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E								
F								

	8	9	A	B	C	D	E	F
C	FCMOVE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVU							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E		FUCOMPP						
F								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.4 Escape Opcodes with DB as First Byte

Table A-13 and A-14 contain maps for escape instruction opcodes that begin with DBH. Table A-13 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-13. DB Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FILD dword-integer	FISTTP dword-integer	FIST dword-integer	FISTP dword-integer		FLD extended-real		FSTP extended-real

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-14 shows the map if the ModR/M byte is outside the range of 00H-BFH. Here, the first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-14. DB Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FCMOVNB							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVNBE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E			FCLEX	FINIT				
F	FCOMI							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

	8	9	A	B	C	D	E	F
C	FCMOVNE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVNU							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E	FUCOMI							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.5 Escape Opcodes with DC as First Byte

Table A-15 and A-16 contain maps for escape instruction opcodes that begin with DCH. Table A-15 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-15. DC Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte (refer to Figure A-1)							
000B	001B	010B	011B	100B	101B	110B	111B
FADD double-real	FMUL double-real	FCOM double-real	FCOMP double-real	FSUB double-real	FSUBR double-real	FDIV double-real	FDIVR double-real

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-16 shows the map if the ModR/M byte is outside the range of 00H-BFH. In this case the first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-16. DC Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FADD							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D								
E	FSUBR							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIVR							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

	8	9	A	B	C	D	E	F
C	FMUL							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D								
E	FSUB							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIV							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.6 Escape Opcodes with DD as First Byte

Table A-17 and A-18 contain maps for escape instruction opcodes that begin with DDH. Table A-17 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-17. DD Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FLD double-real	FISTTP integer64	FST double-real	FSTP double-real	FRSTOR 98/108bytes		FSAVE 98/108bytes	FSTSW 2 bytes

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-18 shows the map if the ModR/M byte is outside the range of 00H-BFH. The first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-18. DD Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FFREE							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
D	FST							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
E	FUCOM							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F								

	8	9	A	B	C	D	E	F
C								
D	FSTP							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
E	FUCOMP							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
F								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.7 Escape Opcodes with DE as First Byte

Table A-19 and A-20 contain opcode maps for escape instruction opcodes that begin with DEH. Table A-19 shows the opcode map if the ModR/M byte is in the range of 00H-BFH. In this case, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-19. DE Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FIADD word-integer	FIMUL word-integer	FICOM word-integer	FICOMP word-integer	FISUB word-integer	FISUBR word-integer	FIDIV word-integer	FIDIVR word-integer

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-20 shows the opcode map if the ModR/M byte is outside the range of 00H-BFH. The first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-20. DE Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FADDP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D								
E	FSUBRP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIVRP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

	8	9	A	B	C	D	E	F
C	FMULP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D		FCOMPP						
E	FSUBP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIVP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.8 Escape Opcodes with DF As First Byte

Table A-21 and A-22 contain the opcode maps for escape instruction opcodes that begin with DFH. Table A-21 shows the opcode map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-21. DF Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FILD word-integer	FISTTP word-integer	FIST word-integer	FISTP word-integer	FBLD packed-BCD	FILD qword-integer	FBSTP packed-BCD	FISTP qword-integer

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-22 shows the opcode map if the ModR/M byte is outside the range of 00H-BFH. The first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-22. DF Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C								
D								
E	FSTSW AX							
F	FCOMIP							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

	8	9	A	B	C	D	E	F
C								
D								
E	FUCOMIP							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

11. Updates to Chapter 2, Volume 3A

Change bars and text show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- In Section 2.1.4, added note clarifying event handling based on IDT event delivery, return using IRET, and Flexible Return and Event Delivery (FRED).
- In Section 2.1.6.1, added reference to MSRs that control FRED transitions.
- Added CR4.FRED in Section 2.5, "Control Registers".
- Updated Table 2-3, "Summary of System Instructions", adding RDMSRLIST, WRMSRLIST, WRMSRNS, SWAPGS, and LKGS.
- In Section 2.2, "Modes of Operation," updated references in Figure 2-3, "Transitions Among the Processor's Operating Modes," to Section 12.8.5 and Section 12.8.5.4.
- In Section 2.5, "Control Registers", Figure 2-7, "Control Registers" was updated to include CR4.FRED at bit 32.

IA-32 architecture (beginning with the Intel386 processor family) provides extensive support for operating-system and system-development software. This support offers multiple modes of operation, which include:

- Real mode, protected mode, virtual 8086 mode, and system management mode. These are sometimes referred to as legacy modes.

Intel 64 architecture supports almost all the system programming facilities available in IA-32 architecture and extends them to a new operating mode (IA-32e mode) that supports a 64-bit programming environment. IA-32e mode allows software to operate in one of two sub-modes:

- 64-bit mode supports 64-bit OS and 64-bit applications
- Compatibility mode allows most legacy software to run; it co-exists with 64-bit applications under a 64-bit OS.

The IA-32 system-level architecture includes features to assist in the following operations:

- Memory management.
- Protection of software modules.
- Multitasking.
- Exception and interrupt handling.
- Multiprocessing.
- Cache management.
- Hardware resource and power management.
- Debugging and performance monitoring.

This chapter provides a description of each part of this architecture. It also describes the system registers that are used to set up and control the processor at the system level and gives a brief overview of the processor's system-level (operating system) instructions.

Many features of the system-level architecture are used only by system programmers. However, application programmers may need to read this chapter and the following chapters in order to create a reliable and secure environment for application programs.

This overview and most subsequent chapters of this book focus on protected-mode operation of the IA-32 architecture. IA-32e mode operation of the Intel 64 architecture, as it differs from protected mode operation, is also described.

All Intel 64 and IA-32 processors enter real-address mode following a power-up or reset (see Chapter 12, "Processor Management and Initialization"). Software then initiates the switch from real-address mode to protected mode. If IA-32e mode operation is desired, software also initiates a switch from protected mode to IA-32e mode.

2.1 OVERVIEW OF THE SYSTEM-LEVEL ARCHITECTURE

System-level architecture consists of a set of registers, data structures, and instructions designed to support basic system-level operations such as memory management, interrupt and exception handling, task management, and control of multiple processors.

Figure 2-1 provides a summary of system registers and data structures that applies to 32-bit modes. System registers and data structures that apply to IA-32e mode are shown in Figure 2-2.

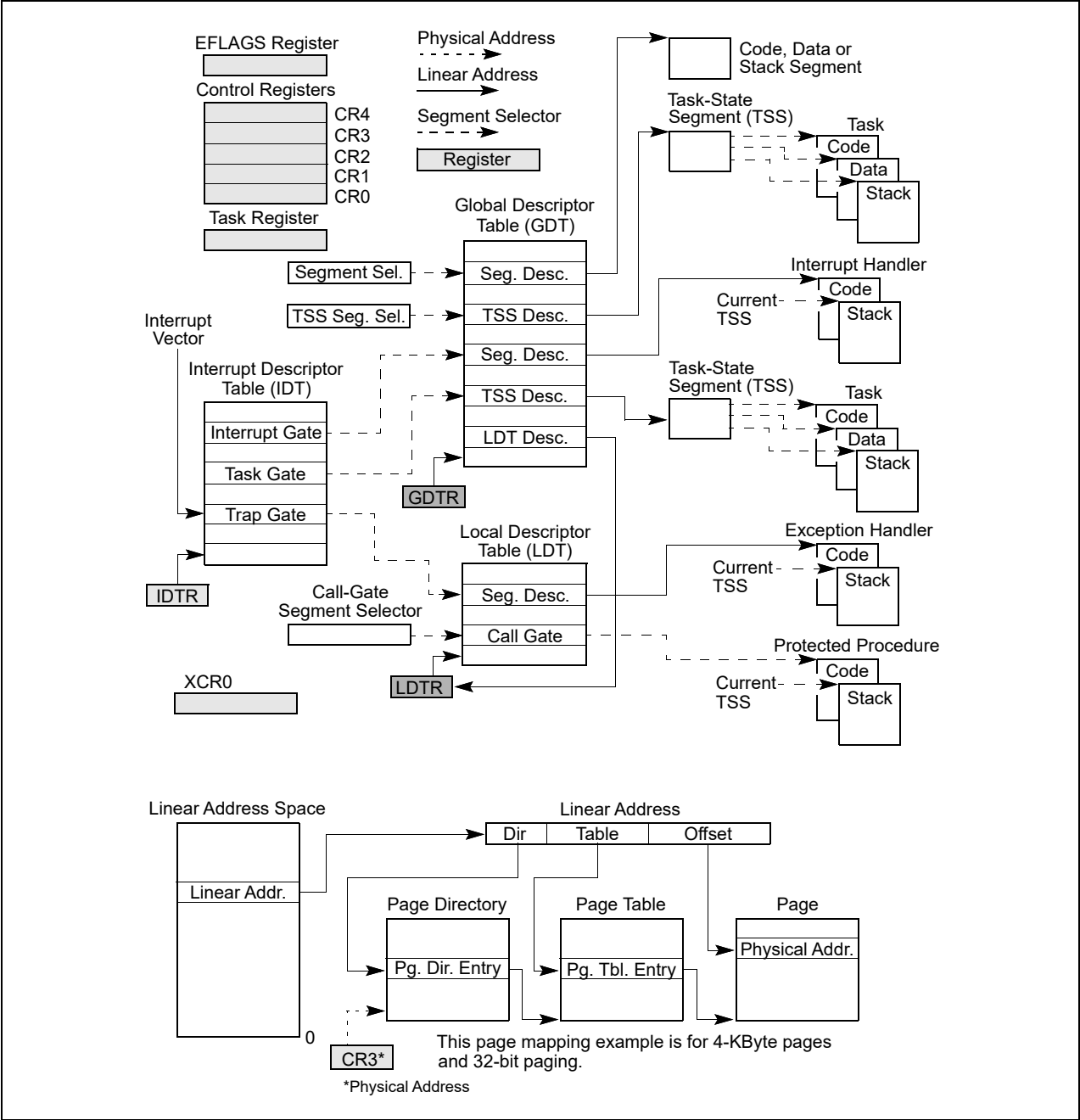


Figure 2-1. IA-32 System-Level Registers and Data Structures

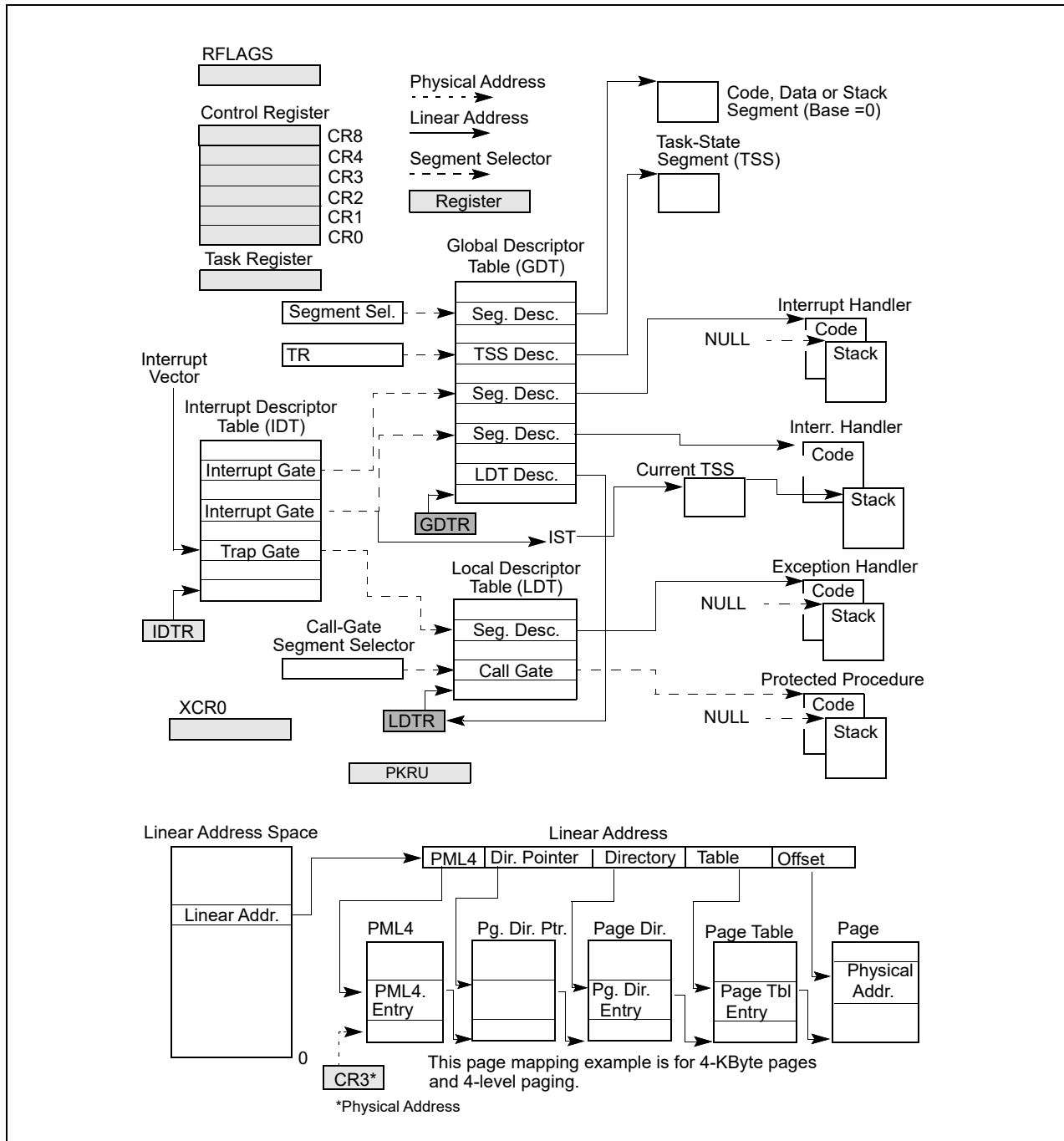


Figure 2-2. System-Level Registers and Data Structures in IA-32e Mode and 4-Level Paging

2.1.1 Global and Local Descriptor Tables

When operating in protected mode, all memory accesses pass through either the global descriptor table (GDT) or an optional local descriptor table (LDT) as shown in Figure 2-1. These tables contain entries called segment descriptors. Segment descriptors provide the base address of segments well as access rights, type, and usage information.

Each segment descriptor has an associated segment selector. A segment selector provides the software that uses it with an index into the GDT or LDT (the offset of its associated segment descriptor), a global/local flag (determines whether the selector points to the GDT or the LDT), and access rights information.

To access a byte in a segment, a segment selector and an offset must be supplied. The segment selector provides access to the segment descriptor for the segment (in the GDT or LDT). From the segment descriptor, the processor obtains the base address of the segment in the linear address space. The offset then provides the location of the byte relative to the base address. This mechanism can be used to access any valid code, data, or stack segment, provided the segment is accessible from the current privilege level (CPL) at which the processor is operating. The CPL is defined as the protection level of the currently executing code segment.

See Figure 2-1. The solid arrows in the figure indicate a linear address, dashed lines indicate a segment selector, and the dotted arrows indicate a physical address. For simplicity, many of the segment selectors are shown as direct pointers to a segment. However, the actual path from a segment selector to its associated segment is always through a GDT or LDT.

The linear address of the base of the GDT is contained in the GDT register (GDTR); the linear address of the LDT is contained in the LDT register (LDTR).

2.1.1.1 Global and Local Descriptor Tables in IA-32e Mode

GDTR and LDTR registers are expanded to 64-bits wide in both IA-32e sub-modes (64-bit mode and compatibility mode). For more information: see Section 3.5.2, "Segment Descriptor Tables in IA-32e Mode."

Global and local descriptor tables are expanded in 64-bit mode to support 64-bit base addresses, (16-byte LDT descriptors hold a 64-bit base address and various attributes). In compatibility mode, descriptors are not expanded.

2.1.2 System Segments, Segment Descriptors, and Gates

Besides code, data, and stack segments that make up the execution environment of a program or procedure, the architecture defines two system segments: the task-state segment (TSS) and the LDT. The GDT is not considered a segment because it is not accessed by means of a segment selector and segment descriptor. TSSs and LDTs have segment descriptors defined for them.

The architecture also defines a set of special descriptors called gates (call gates, interrupt gates, trap gates, and task gates). These provide protected gateways to system procedures and handlers that may operate at a different privilege level than application programs and most procedures. For example, a CALL to a call gate can provide access to a procedure in a code segment that is at the same or a numerically lower privilege level (more privileged) than the current code segment. To access a procedure through a call gate, the calling procedure¹ supplies the selector for the call gate. The processor then performs an access rights check on the call gate, comparing the CPL with the privilege level of the call gate and the destination code segment pointed to by the call gate.

If access to the destination code segment is allowed, the processor gets the segment selector for the destination code segment and an offset into that code segment from the call gate. If the call requires a change in privilege level, the processor also switches to the stack for the targeted privilege level. The segment selector for the new stack is obtained from the TSS for the currently running task. Gates also facilitate transitions between 16-bit and 32-bit code segments, and vice versa.

2.1.2.1 Gates in IA-32e Mode

In IA-32e mode, the following descriptors are 16-byte descriptors (expanded to allow a 64-bit base): LDT descriptors, 64-bit TSSs, call gates, interrupt gates, and trap gates.

Call gates facilitate transitions between 64-bit mode and compatibility mode. Task gates are not supported in IA-32e mode. On privilege level changes, stack segment selectors are not read from the TSS. Instead, they are set to NULL.

1. The word "procedure" is commonly used in this document as a general term for a logical unit or block of code (such as a program, procedure, function, or routine).

2.1.3 Task-State Segments and Task Gates

The TSS (see Figure 2-1) defines the state of the execution environment for a task. It includes the state of general-purpose registers, segment registers, the EFLAGS register, the EIP register, and segment selectors with stack pointers for three stack segments (one stack for each privilege level). The TSS also includes the segment selector for the LDT associated with the task and the base address of the paging-structure hierarchy.

All program execution in protected mode happens within the context of a task (called the current task). The segment selector for the TSS for the current task is stored in the task register. The simplest method for switching to a task is to make a call or jump to the new task. Here, the segment selector for the TSS of the new task is given in the CALL or JMP instruction. In switching tasks, the processor performs the following actions:

1. Stores the state of the current task in the current TSS.
2. Loads the task register with the segment selector for the new task.
3. Accesses the new TSS through a segment descriptor in the GDT.
4. Loads the state of the new task from the new TSS into the general-purpose registers, the segment registers, the LDTR, control register CR3 (base address of the paging-structure hierarchy), the EFLAGS register, and the EIP register.
5. Begins execution of the new task.

A task can also be accessed through a task gate. A task gate is similar to a call gate, except that it provides access (through a segment selector) to a TSS rather than a code segment.

2.1.3.1 Task-State Segments in IA-32e Mode

Hardware task switches are not supported in IA-32e mode. However, TSSs continue to exist. The base address of a TSS is specified by its descriptor.

A 64-bit TSS holds the following information that is important to 64-bit operation:

- Stack pointer addresses for each privilege level.
- Pointer addresses for the interrupt stack table.
- Offset address of the IO-permission bitmap (from the TSS base).

The task register is expanded to hold 64-bit base addresses in IA-32e mode. See also: Section 10.7, “Task Management in 64-bit Mode.”

2.1.4 Interrupt and Exception Handling

NOTE

This section focuses on event handling based on IDT event delivery and on return using the IRET instruction. FRED transitions are alternatives to these mechanisms. In general, the material in this section does not apply when FRED transitions are enabled. See Chapter 8, “Flexible Return and Event Delivery (FRED),” for more information about FRED transitions.

External interrupts, software interrupts and exceptions are handled through the interrupt descriptor table (IDT). The IDT stores a collection of gate descriptors that provide access to interrupt and exception handlers. Like the GDT, the IDT is not a segment. The linear address for the base of the IDT is contained in the IDT register (IDTR).

Gate descriptors in the IDT can be interrupt, trap, or task gate descriptors. To access an interrupt or exception handler, the processor first receives an interrupt vector from internal hardware, an external interrupt controller, or from software by means of an INT *n*, INTO, INT3, INT1, or BOUND instruction. The interrupt vector provides an index into the IDT. If the selected gate descriptor is an interrupt gate or a trap gate, the associated handler procedure is accessed in a manner similar to calling a procedure through a call gate. If the descriptor is a task gate, the handler is accessed through a task switch.

2.1.4.1 Interrupt and Exception Handling IA-32e Mode

In IA-32e mode, interrupt gate descriptors are expanded to 16 bytes to support 64-bit base addresses. This is true for 64-bit mode and compatibility mode.

The IDTR register is expanded to hold a 64-bit base address. Task gates are not supported.

2.1.5 Memory Management

System architecture supports either direct physical addressing of memory or virtual memory (through paging). When physical addressing is used, a linear address is treated as a physical address. When paging is used: all code, data, stack, and system segments (including the GDT and IDT) can be paged with only the most recently accessed pages being held in physical memory.

The location of pages (sometimes called page frames) in physical memory is contained in the paging structures. These structures reside in physical memory (see Figure 2-1 for the case of 32-bit paging).

The base physical address of the paging-structure hierarchy is contained in control register CR3. The entries in the paging structures determine the physical address of the base of a page frame, access rights and memory management information.

To use this paging mechanism, a linear address is broken into parts. The parts provide separate offsets into the paging structures and the page frame. A system can have a single hierarchy of paging structures or several. For example, each task can have its own hierarchy.

2.1.5.1 Memory Management in IA-32e Mode

In IA-32e mode, physical memory pages are managed by a set of system data structures. In both compatibility mode and 64-bit mode, four or five levels of system data structures are used (see Chapter 5, "Paging"). These include the following:

- **The page map level 5 (PML5)** — An entry in the PML5 table contains the physical address of the base of a PML4 table, access rights, and memory management information. The base physical address of the PML5 table is stored in CR3. The PML5 table is used only with 5-level paging.
- **A page map level 4 (PML4)** — An entry in a PML4 table contains the physical address of the base of a page directory pointer table, access rights, and memory management information. With 4-level paging, there is only one PML4 table and its base physical address is stored in CR3.
- **A set of page directory pointer tables** — An entry in a page directory pointer table contains the physical address of the base of a page directory table, access rights, and memory management information.
- **Sets of page directories** — An entry in a page directory table contains the physical address of the base of a page table, access rights, and memory management information.
- **Sets of page tables** — An entry in a page table contains the physical address of a page frame, access rights, and memory management information.

2.1.6 System Registers

To assist in initializing the processor and controlling system operations, the system architecture provides system flags in the EFLAGS register and several system registers:

- The system flags and IOPL field in the EFLAGS register control task and mode switching, interrupt handling, instruction tracing, and access rights. See also: Section 2.3, "System Flags and Fields in the EFLAGS Register."
- The control registers (CR0, CR2, CR3, and CR4) contain a variety of flags and data fields for controlling system-level operations. Other flags in these registers are used to indicate support for specific processor capabilities within the operating system or executive. See also: Chapter 2, "Control Registers," and Section 2.6, "Extended Control Registers (Including XCR0)."
- The debug registers (not shown in Figure 2-1) allow the setting of breakpoints for use in debugging programs and systems software. See also: Chapter 20, "Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features."

- The GDTR, LDTR, and IDTR registers contain the linear addresses and sizes (limits) of their respective tables. See also: Section 2.4, “Memory-Management Registers.”
- The task register contains the linear address and size of the TSS for the current task. See also: Section 2.4, “Memory-Management Registers.”
- Model-specific registers (not shown in Figure 2-1).

The model-specific registers (MSRs) are a group of registers available primarily to operating-system or executive procedures (that is, code running at privilege level 0). These registers control items such as the debug extensions, the performance-monitoring counters, the machine-check architecture, and the memory type ranges (MTRRs).

The number and function of these registers varies among different members of the Intel 64 and IA-32 processor families. See also: Section 12.4, “Model-Specific Registers (MSRs),” and Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

Most systems restrict access to system registers (other than the EFLAGS register) by application programs. Systems can be designed, however, where all programs and procedures run at the most privileged level (privilege level 0). In such a case, application programs would be allowed to modify the system registers.

2.1.6.1 System Registers in IA-32e Mode

In IA-32e mode, the four system-descriptor-table registers (GDTR, IDTR, LDTR, and TR) are expanded in hardware to hold 64-bit base addresses. EFLAGS becomes the 64-bit RFLAGS register. CR0–CR4 are expanded to 64 bits. CR8 becomes available. CR8 provides read-write access to the task priority register (TPR) so that the operating system can control the priority classes of external interrupts.

In 64-bit mode, debug registers DR0–DR7 are 64 bits. In compatibility mode, address-matching in DR0–DR3 is also done at 64-bit granularity.

On systems that support IA-32e mode, the extended feature enable register (IA32_EFER) is available. This model-specific register controls activation of IA-32e mode and other IA-32e mode operations. In addition, there are several model-specific registers that govern IA-32e mode instructions:

- **IA32_KERNEL_GS_BASE** — Used by SWAPGS instruction.
- **IA32_LSTAR** — Used by SYSCALL instruction.
- **IA32_FMASK** — Used by SYSCALL instruction.
- **IA32_STAR** — Used by SYSCALL and SYSRET instruction.

Another set of MSRs control FRED transitions when they are enabled. These are described in Section 8.2.

2.1.7 Other System Resources

Besides the system registers and data structures described in the previous sections, system architecture provides the following additional resources:

- Operating system instructions (see also: Section 2.8, “System Instruction Summary”).
- Performance-monitoring counters (not shown in Figure 2-1).
- Internal caches and buffers (not shown in Figure 2-1).

Performance-monitoring counters are event counters that can be programmed to count processor events such as the number of instructions decoded, the number of interrupts received, or the number of cache loads.

The processor provides several internal caches and buffers. The caches are used to store both data and instructions. The buffers are used to store things like decoded addresses to system and application segments and write operations waiting to be performed. See also: Chapter 14, “Memory Cache Control.”

2.2 MODES OF OPERATION

The IA-32 architecture supports three operating modes and one quasi-operating mode:

- **Protected mode** — This is the native operating mode of the processor. It provides a rich set of architectural features, flexibility, high performance and backward compatibility to existing software base.
- **Real-address mode** — This operating mode provides the programming environment of the Intel 8086 processor, with a few extensions (such as the ability to switch to protected or system management mode).
- **System management mode (SMM)** — SMM is a standard architectural feature in all IA-32 processors, beginning with the Intel386 SL processor. This mode provides an operating system or executive with a transparent mechanism for implementing power management and OEM differentiation features. SMM is entered through activation of an external system interrupt pin (SMI#), which generates a system management interrupt (SMI). In SMM, the processor switches to a separate address space while saving the context of the currently running program or task. SMM-specific code may then be executed transparently. Upon returning from SMM, the processor is placed back into its state prior to the SMI.
- **Virtual-8086 mode** — In protected mode, the processor supports a quasi-operating mode known as virtual-8086 mode. This mode allows the processor execute 8086 software in a protected, multitasking environment.

Intel 64 architecture supports all operating modes of IA-32 architecture and IA-32e modes:

- **IA-32e mode** — In IA-32e mode, the processor supports two sub-modes: compatibility mode and 64-bit mode. 64-bit mode provides 64-bit linear addressing and support for physical address space larger than 64 GBytes. Compatibility mode allows most legacy protected-mode applications to run unchanged.

Figure 2-3 shows how the processor moves between operating modes.

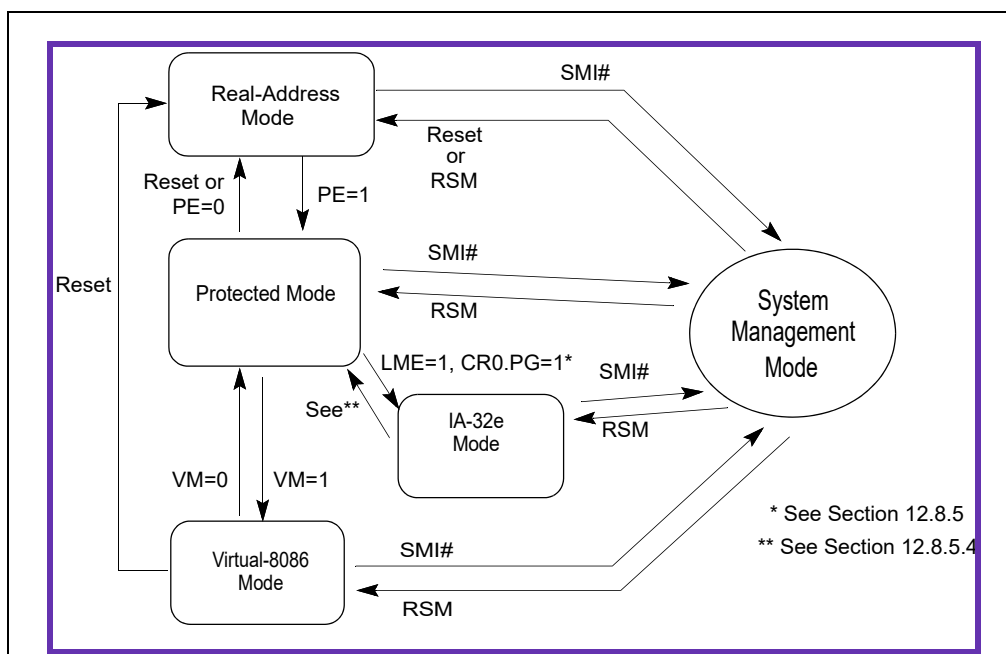


Figure 2-3. Transitions Among the Processor's Operating Modes

The processor is placed in real-address mode following power-up or a reset. The PE flag in control register CR0 then controls whether the processor is operating in real-address or protected mode. See also: Section 12.9, "Mode Switching," and Section 5.1.2, "Paging-Mode Enabling."

The VM flag in the EFLAGS register determines whether the processor is operating in protected mode or virtual-8086 mode. Transitions between protected mode and virtual-8086 mode are generally carried out as part of a task switch or a return from an interrupt or exception handler. See also: Section 23.2.5, "Entering Virtual-8086 Mode."

The LMA bit (IA32_EFER.LMA[10]) determines whether the processor is operating in IA-32e mode. When running in IA-32e mode, 64-bit or compatibility sub-mode operation is determined by CS.L bit of the code segment. The processor enters into IA-32e mode from protected mode by enabling paging and setting the LME bit (IA32_EFER.LME[8]). See also: Chapter 12, "Processor Management and Initialization."

The processor switches to SMM whenever it receives an SMI while the processor is in real-address, protected, virtual-8086, or IA-32e modes. Upon execution of the RSM instruction, the processor always returns to the mode it was in when the SMI occurred.

2.2.1 Extended Feature Enable Register

The IA32_EFER MSR provides several fields related to IA-32e mode enabling and operation. It also provides one field that relates to page-access right modification (see Section 5.6, “Access Rights”). The layout of the IA32_EFER MSR is shown in Figure 2-4.

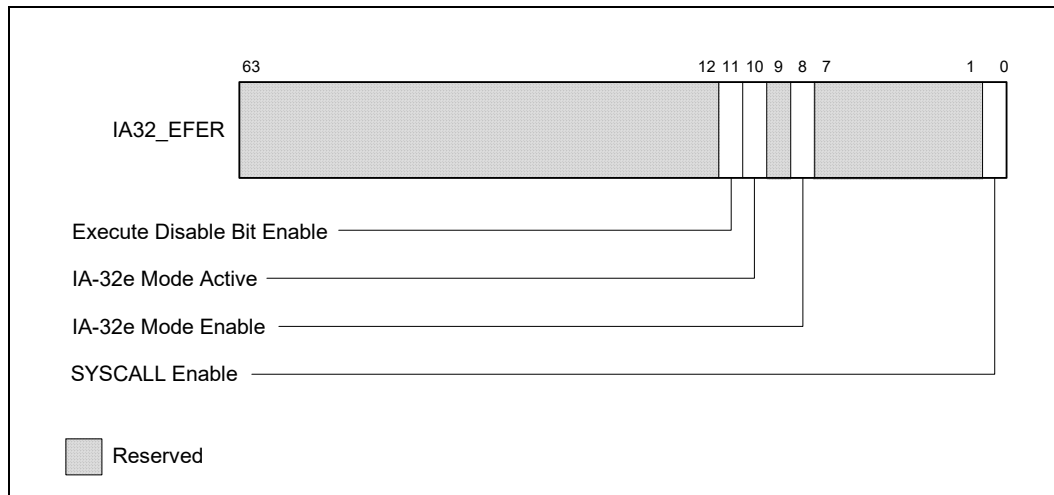


Figure 2-4. IA32_EFER MSR Layout

Table 2-1. IA32_EFER MSR Information

Bit	Description
0	SYSCALL Enable: IA32_EFER.SCE (R/W) Enables SYSCALL/SYSRET instructions in 64-bit mode.
7:1	Reserved.
8	IA-32e Mode Enable: IA32_EFER.LME (R/W) Enables IA-32e mode operation.
9	Reserved.
10	IA-32e Mode Active: IA32_EFER.LMA (R) Indicates IA-32e mode is active when set.
11	Execute Disable Bit Enable: IA32_EFER.NXE (R/W) Enables page access restriction by preventing instruction fetches from PAE pages with the XD bit set (See Section 5.6).
63:12	Reserved.

2.3 SYSTEM FLAGS AND FIELDS IN THE EFLAGS REGISTER

The system flags and IOPL field of the EFLAGS register control I/O, maskable hardware interrupts, debugging, task switching, and the virtual-8086 mode (see Figure 2-5). Only privileged code (typically operating system or executive code) should be allowed to modify these bits.

The system flags and IOPL are:

TF **Trap (bit 8)** — Set to enable single-step mode for debugging; clear to disable single-step mode. In single-step mode, the processor generates a debug exception after each instruction. This allows the execution state of a program to be inspected after each instruction. If an application program sets the TF flag using a POPF, POPFD, or IRET instruction, a debug exception is generated after the instruction that follows the POPF, POPFD, or IRET.

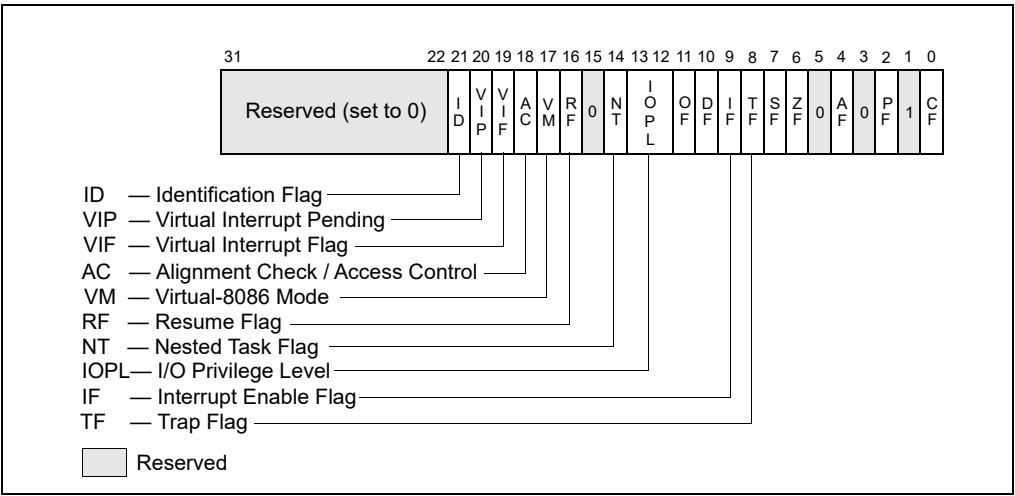


Figure 2-5. System Flags in the EFLAGS Register

IF **Interrupt enable (bit 9)** — Controls the response of the processor to maskable hardware interrupt requests (see also: Section 7.3.2, “Maskable Hardware Interrupts”). The flag is set to respond to maskable hardware interrupts; cleared to inhibit maskable hardware interrupts. The IF flag does not affect the generation of exceptions or nonmaskable interrupts (NMI interrupts). The CPL, IOPL, and the state of the VME flag in control register CR4 determine whether the IF flag can be modified by the CLI, STI, POPF, POPFD, and IRET.

IOPL **I/O privilege level field (bits 12 and 13)** — Indicates the I/O privilege level (IOPL) of the currently running program or task. The CPL of the currently running program or task must be less than or equal to the IOPL to access the I/O address space. The POPF and IRET instructions can modify this field only when operating at a CPL of 0.

The IOPL is also one of the mechanisms that controls the modification of the IF flag and the handling of interrupts in virtual-8086 mode when virtual mode extensions are in effect (when CR4.VME = 1). See also: Chapter 20, “Input/Output,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

NT **Nested task (bit 14)** — Controls the chaining of interrupted and called tasks. The processor sets this flag on calls to a task initiated with a CALL instruction, an interrupt, or an exception. It examines and modifies this flag on returns from a task initiated with the IRET instruction. The flag can be explicitly set or cleared with the POPF/POPF instructions; however, changing to the state of this flag can generate unexpected exceptions in application programs.

See also: Section 10.4, “Task Linking.”

RF **Resume (bit 16)** — Controls the processor’s response to instruction-breakpoint conditions. When set, this flag temporarily disables debug exceptions (#DB) from being generated for instruction breakpoints (although other exception conditions can cause an exception to be generated). When clear, instruction breakpoints will generate debug exceptions.

The primary function of the RF flag is to allow the restarting of an instruction following a debug exception that was caused by an instruction breakpoint condition. Here, debug software must set this flag in the EFLAGS image on the stack just prior to returning to the interrupted program with IRETD (to prevent the instruction breakpoint from causing another debug exception). The processor then automatically clears

this flag after the instruction returned to has been successfully executed, enabling instruction breakpoint faults again.

See also: Section 20.3.1.1, “Instruction-Breakpoint Exception Condition.”

VM Virtual-8086 mode (bit 17) — Set to enable virtual-8086 mode; clear to return to protected mode.

See also: Section 23.2.1, “Enabling Virtual-8086 Mode.”

AC Alignment check or access control (bit 18) — If the AM bit is set in the CR0 register, alignment checking of user-mode data accesses is enabled if and only if this flag is 1. An alignment-check exception is generated when reference is made to an unaligned operand, such as a word at an odd byte address or a doubleword at an address which is not an integral multiple of four. Alignment-check exceptions are generated only in user mode (privilege level 3). Memory references that default to privilege level 0, such as segment descriptor loads, do not generate this exception even when caused by instructions executed in user-mode.

The alignment-check exception can be used to check alignment of data. This is useful when exchanging data with processors which require all data to be aligned. The alignment-check exception can also be used by interpreters to flag some pointers as special by misaligning the pointer. This eliminates overhead of checking each pointer and only handles the special pointer when used.

If the SMAP bit is set in the CR4 register, explicit supervisor-mode data accesses to user-mode pages are allowed if and only if this bit is 1. See Section 5.6, “Access Rights.”

VIF Virtual Interrupt (bit 19) — Contains a virtual image of the IF flag. This flag is used in conjunction with the VIP flag. The processor only recognizes the VIF flag when either the VME flag or the PVI flag in control register CR4 is set and the IOPL is less than 3. (The VME flag enables the virtual-8086 mode extensions; the PVI flag enables the protected-mode virtual interrupts.)

See also: Section 23.3.3.5, “Method 6: Software Interrupt Handling,” and Section 23.4, “Protected-Mode Virtual Interrupts.”

VIP Virtual interrupt pending (bit 20) — Set by software to indicate that an interrupt is pending; cleared to indicate that no interrupt is pending. This flag is used in conjunction with the VIF flag. The processor reads this flag but never modifies it. The processor only recognizes the VIP flag when either the VME flag or the PVI flag in control register CR4 is set and the IOPL is less than 3. The VME flag enables the virtual-8086 mode extensions; the PVI flag enables the protected-mode virtual interrupts.

See Section 23.3.3.5, “Method 6: Software Interrupt Handling,” and Section 23.4, “Protected-Mode Virtual Interrupts.”

ID Identification (bit 21) — The ability of a program or procedure to set or clear this flag indicates support for the CPUID instruction.

2.3.1 System Flags and Fields in IA-32e Mode

In 64-bit mode, the RFLAGS register expands to 64 bits with the upper 32 bits reserved. System flags in RFLAGS (64-bit mode) or EFLAGS (compatibility mode) are shown in Figure 2-5.

In IA-32e mode, the processor does not allow the VM bit to be set because virtual-8086 mode is not supported (attempts to set the bit are ignored). Also, the processor will not set the NT bit. The processor does, however, allow software to set the NT bit (note that an IRET causes a general protection fault in IA-32e mode if the NT bit is set).

In IA-32e mode, the SYSCALL/SYSRET instructions have a programmable method of specifying which bits are cleared in RFLAGS/EFLAGS. These instructions save/restore EFLAGS/RFLAGS.

2.4 MEMORY-MANAGEMENT REGISTERS

The processor provides four memory-management registers (GDTR, LDTR, IDTR, and TR) that specify the locations of the data structures which control segmented memory management (see Figure 2-6). Special instructions are provided for loading and storing these registers.

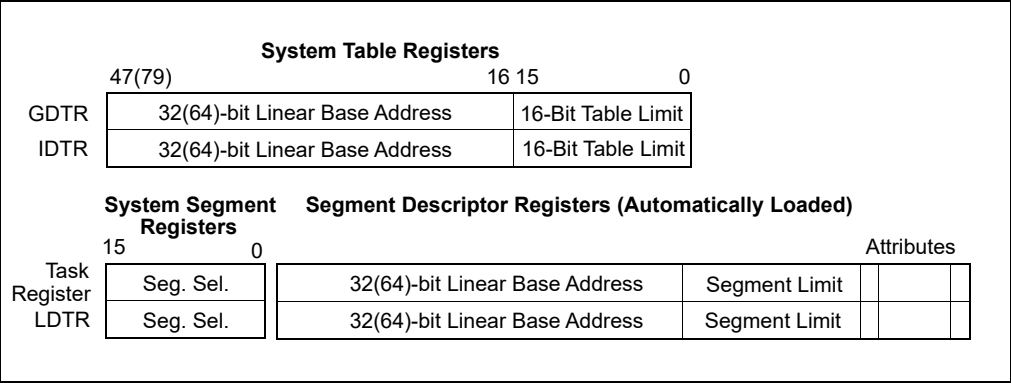


Figure 2-6. Memory Management Registers

2.4.1 Global Descriptor Table Register (GDTR)

The GDTR register holds the base address (32 bits in protected mode; 64 bits in IA-32e mode) and the 16-bit table limit for the GDT. The base address specifies the linear address of byte 0 of the GDT; the table limit specifies the number of bytes in the table.

The LGDT and SGDT instructions load and store the GDTR register, respectively. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH. A new base address must be loaded into the GDTR as part of the processor initialization process for protected-mode operation.

See also: Section 3.5.1, "Segment Descriptor Tables."

2.4.2 Local Descriptor Table Register (LDTR)

The LDTR register holds the 16-bit segment selector, base address (32 bits in protected mode; 64 bits in IA-32e mode), segment limit, and descriptor attributes for the LDT. The base address specifies the linear address of byte 0 of the LDT segment; the segment limit specifies the number of bytes in the segment. See also: Section 3.5.1, "Segment Descriptor Tables."

The LLDT and SLDT instructions load and store the segment selector part of the LDTR register, respectively. The segment that contains the LDT must have a segment descriptor in the GDT. When the LLDT instruction loads a segment selector in the LDTR: the base address, limit, and descriptor attributes from the LDT descriptor are automatically loaded in the LDTR.

When a task switch occurs, the LDTR is automatically loaded with the segment selector and descriptor for the LDT for the new task. The contents of the LDTR are not automatically saved prior to writing the new LDT information into the register.

On power up or reset of the processor, the segment selector and base address are set to the default value of 0 and the limit is set to 0FFFFH.

2.4.3 IDTR Interrupt Descriptor Table Register

The IDTR register holds the base address (32 bits in protected mode; 64 bits in IA-32e mode) and 16-bit table limit for the IDT. The base address specifies the linear address of byte 0 of the IDT; the table limit specifies the number of bytes in the table. The LIDT and SIDT instructions load and store the IDTR register, respectively. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH. The base address and limit in the register can then be changed as part of the processor initialization process.

See also: Section 7.10, "Interrupt Descriptor Table (IDT)."

2.4.4 Task Register (TR)

The task register holds the 16-bit segment selector, base address (32 bits in protected mode; 64 bits in IA-32e mode), segment limit, and descriptor attributes for the TSS of the current task. The selector references the TSS descriptor in the GDT. The base address specifies the linear address of byte 0 of the TSS; the segment limit specifies the number of bytes in the TSS. See also: Section 10.2.4, “Task Register.”

The LTR and STR instructions load and store the segment selector part of the task register, respectively. When the LTR instruction loads a segment selector in the task register, the base address, limit, and descriptor attributes from the TSS descriptor are automatically loaded into the task register. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH.

When a task switch occurs, the task register is automatically loaded with the segment selector and descriptor for the TSS for the new task. The contents of the task register are not automatically saved prior to writing the new TSS information into the register.

2.5 CONTROL REGISTERS

Control registers (CR0, CR1, CR2, CR3, and CR4; see Figure 2-7) determine operating mode of the processor and the characteristics of the currently executing task. These registers are 32 bits in all 32-bit modes and compatibility mode.

In 64-bit mode, control registers are expanded to 64 bits. The MOV CRn instructions are used to manipulate the register bits. Operand-size prefixes for these instructions are ignored. The following is also true:

- The control registers can be read and loaded (or modified) using the move-to-or-from-control-registers forms of the MOV instruction. In protected mode, the MOV instructions allow the control registers to be read or loaded (at privilege level 0 only). This restriction means that application programs or operating-system procedures (running at privilege levels 1, 2, or 3) are prevented from reading or loading the control registers.
- Some of the bits in CR0 and CR4 are reserved and must be written with zeros. Attempting to set any reserved bits in CR0[31:0] is ignored. Attempting to set any reserved bits in CR0[63:32] results in a general-protection exception, #GP(0). Attempting to set any reserved bits in CR4 results in a general-protection exception, #GP(0).
- All 64 bits of CR2 are writable by software.
- Bits in CR3 in the range 63:MAXPHYADDR that are reserved (see Figure 2-7) must be zero. Attempting to set any of them results in #GP(0).¹
- The MOV CR2 instruction does not check that address written to CR2 is canonical.
- A 64-bit capable processor will retain the upper 32 bits of each control register when transitioning out of IA-32e mode.
- On a 64-bit capable processor, an execution of MOV to CR outside of 64-bit mode zeros the upper 32 bits of the control register.
- Register CR8 is available in 64-bit mode only.

The control registers are summarized below, and each architecturally defined control field in these control registers is described individually. In Figure 2-7, the width of the register in 64-bit mode is indicated in parenthesis (except for CR0).

- **CR0** — Contains system control flags that control operating mode and states of the processor.
- **CR1** — Reserved.
- **CR2** — Contains the page-fault linear address (the linear address that caused a page fault).
- **CR3** — Contains the physical address of the base of the paging-structure hierarchy and four flags (PWT, PCD, LAM_U57, and LAM_U48). Only the most-significant bits (less the lower 12 bits) of the base address are specified; the lower 12 bits of the address are assumed to be 0. The first paging structure must thus be aligned

1. MAXPHYADDR is derived from the value enumerated by CPUID.80000008H:EAX[7:0]. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is out-side secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.

to a page (4-KByte) boundary. The PCD and PWT flags control caching of that paging structure in the processor's internal data caches (they do not control TLB caching of page-directory information). The LAM_U57 and LAM_U48 flags control linear-address masking for user addresses (see Section 4.4, "Linear-Address Masking").

When using the physical address extension, the CR3 register contains the base address of the page-directory-pointer table. With 4-level paging and 5-level paging, the CR3 register contains the base address of the PML4 table and PML5 table, respectively. If PCIDs are enabled, CR3 has a format different from that illustrated in Figure 2-7. See Section 5.5, "4-Level Paging and 5-Level Paging."

See also: Chapter 5, "Paging."

- **CR4** — Contains a group of flags that enable several architectural extensions, and indicate operating system or executive support for specific processor capabilities. Bits CR4[63:32] can only be used for IA-32e mode only features that are enabled after entering 64-bit mode. Bits CR4[31:0] do not have any effect outside of IA-32e mode.
- **CR8** — Provides (in CR8[3:0]) read and write access to bits 7:4 of the local APIC's Task-Priority Register (TPR). These bits are the **task-priority class**; all interrupts in this and lower priority classes are blocked. A value of 15 blocks interrupts in all priority classes, while a value of 0 blocks none. CR8 can be accessed only in 64-bit mode, but the value of the TPR blocks interrupts regardless of mode.

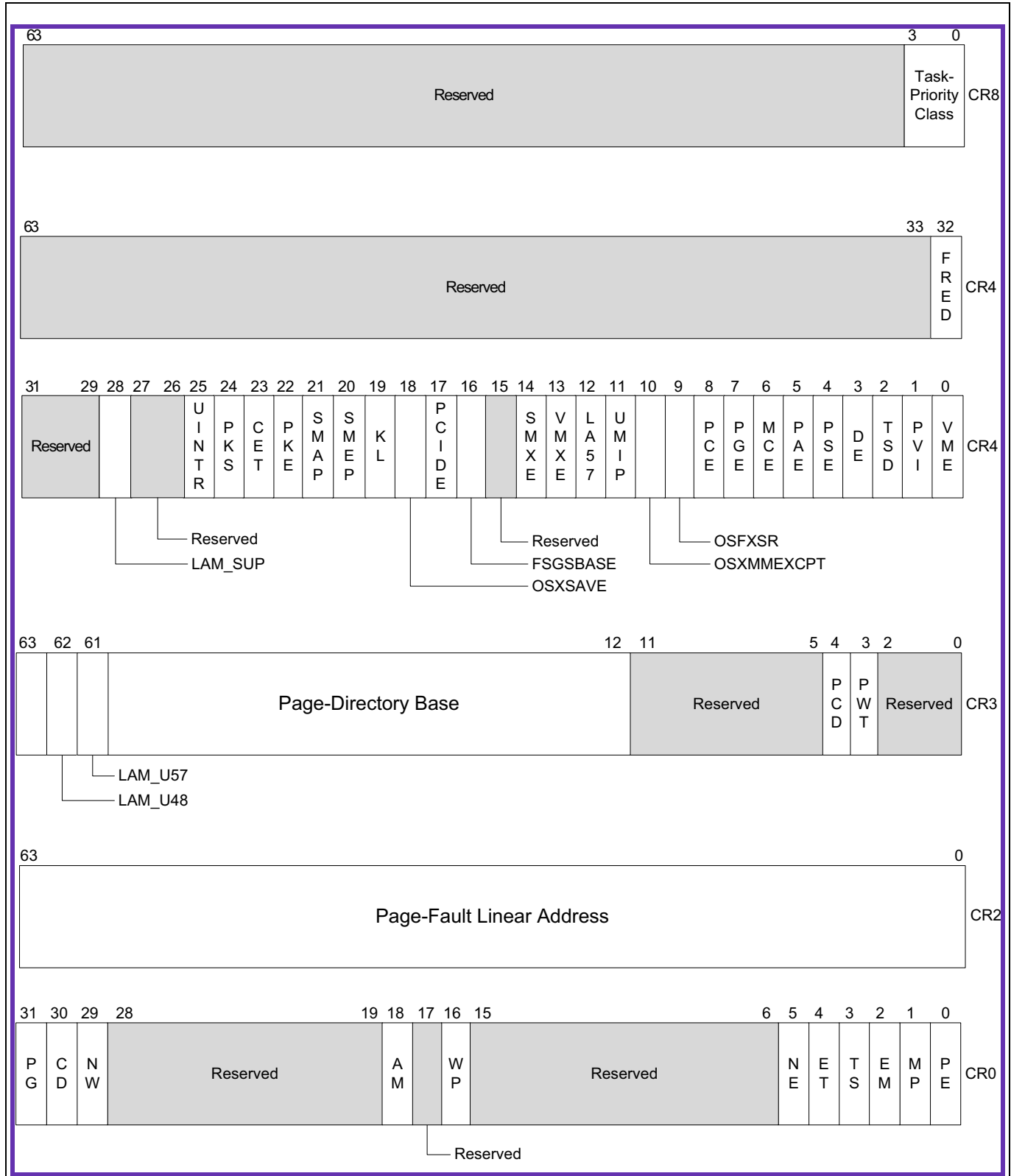


Figure 2-7. Control Registers

The flags in control registers are:

CR0.PG

Paging (bit 31 of CR0) — Enables paging when set; disables paging when clear. When paging is disabled, all linear addresses are treated as physical addresses. The PG flag has no effect if the PE flag (bit 0 of register CR0) is not also set; setting the PG flag when the PE flag is clear causes a general-protection exception (#GP). See also: Chapter 5, “Paging.”

On Intel 64 processors, enabling and disabling IA-32e mode operation also requires modifying CR0.PG.

CR0.CD

Cache Disable (bit 30 of CR0) — When the CD and NW flags are clear, caching of memory locations for the whole of physical memory in the processor’s internal (and external) caches is enabled. When the CD flag is set, caching is restricted as described in Table 14-5. To prevent the processor from accessing and updating its caches, the CD flag must be set and the caches must be invalidated so that no cache hits can occur.

See also: Section 14.5.3, “Preventing Caching,” and Section 14.5, “Cache Control.”

CR0.NW

Not Write-through (bit 29 of CR0) — When the NW and CD flags are clear, write-back (for Pentium 4, Intel Xeon, P6 family, and Pentium processors) or write-through (for Intel486 processors) is enabled for writes that hit the cache and invalidation cycles are enabled. See Table 14-5 for detailed information about the effect of the NW flag on caching for other settings of the CD and NW flags.

CR0.AM

Alignment Mask (bit 18 of CR0) — Enables automatic alignment checking when set; disables alignment checking when clear. Alignment checking is performed only when the AM flag is set, the AC flag in the EFLAGS register is set, CPL is 3, and the processor is operating in either protected or virtual-8086 mode.

CR0.WP

Write Protect (bit 16 of CR0) — When set, inhibits supervisor-level procedures from writing into read-only pages; when clear, allows supervisor-level procedures to write into read-only pages (regardless of the U/S bit setting; see Section 5.1.3 and Section 5.6). This flag facilitates implementation of the copy-on-write method of creating a new process (forking) used by operating systems such as UNIX. This flag must be set before software can set CR4.CET, and it cannot be cleared as long as CR4.CET = 1 (see below).

CR0.NE

Numeric Error (bit 5 of CR0) — Enables the native (internal) mechanism for reporting x87 FPU errors when set; enables the PC-style x87 FPU error reporting mechanism when clear. When the NE flag is clear and the IGNNE# input is asserted, x87 FPU errors are ignored. When the NE flag is clear and the IGNNE# input is deasserted, an unmasked x87 FPU error causes the processor to assert the FERR# pin to generate an external interrupt and to stop instruction execution immediately before executing the next waiting floating-point instruction or WAIT/FWAIT instruction.

The FERR# pin is intended to drive an input to an external interrupt controller (the FERR# pin emulates the ERROR# pin of the Intel 287 and Intel 387 DX math coprocessors). The NE flag, IGNNE# pin, and FERR# pin are used with external logic to implement PC-style error reporting. Using FERR# and IGNNE# to handle floating-point exceptions is deprecated by modern operating systems; this non-native approach also limits newer processors to operate with one logical processor active.

See also: Section 8.7, “Handling x87 FPU Exceptions in Software,” in Chapter 8, “Programming with the x87 FPU,” and Appendix A, “EFLAGS Cross-Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

CR0.ET

Extension Type (bit 4 of CR0) — Reserved in the Pentium 4, Intel Xeon, P6 family, and Pentium processors. In the Pentium 4, Intel Xeon, and P6 family processors, this flag is hardcoded to 1. In the Intel386 and Intel486 processors, this flag indicates support of Intel 387 DX math coprocessor instructions when set.

CR0.TS

Task Switched (bit 3 of CR0) — Allows the saving of the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context on a task switch to be delayed until an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction is

actually executed by the new task. The processor sets this flag on every task switch and tests it when executing x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

- If the TS flag is set and the EM flag (bit 2 of CR0) is clear, a device-not-available exception (#NM) is raised prior to the execution of any x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction; with the exception of PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. See the paragraph below for the special case of the WAIT/FWAIT instructions.
- If the TS flag is set and the MP flag (bit 1 of CR0) and EM flag are clear, an #NM exception is not raised prior to the execution of an x87 FPU WAIT/FWAIT instruction.
- If the EM flag is set, the setting of the TS flag has no effect on the execution of x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

Table 2-2 shows the actions taken when the processor encounters an x87 FPU instruction based on the settings of the TS, EM, and MP flags. Table 15-1 and 16-1 show the actions taken when the processor encounters an MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction.

The processor does not automatically save the context of the x87 FPU, XMM, and MXCSR registers on a task switch. Instead, it sets the TS flag, which causes the processor to raise an #NM exception whenever it encounters an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction in the instruction stream for the new task (with the exception of the instructions listed above).

The fault handler for the #NM exception can then be used to clear the TS flag (with the CLTS instruction) and save the context of the x87 FPU, XMM, and MXCSR registers. If the task never encounters an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction, the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context is never saved.

Table 2-2. Action Taken By x87 FPU Instructions for Different Combinations of EM, MP, and TS

CR0 Flags			x87 FPU Instruction Type	
EM	MP	TS	Floating-Point	WAIT/FWAIT
0	0	0	Execute	Execute.
0	0	1	#NM Exception	Execute.
0	1	0	Execute	Execute.
0	1	1	#NM Exception	#NM exception.
1	0	0	#NM Exception	Execute.
1	0	1	#NM Exception	Execute.
1	1	0	#NM Exception	Execute.
1	1	1	#NM Exception	#NM exception.

CR0.EM

Emulation (bit 2 of CR0) — Indicates that the processor does not have an internal or external x87 FPU when set; indicates an x87 FPU is present when clear. This flag also affects the execution of MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

When the EM flag is set, execution of an x87 FPU instruction generates a device-not-available exception (#NM). This flag must be set when the processor does not have an internal x87 FPU or is not connected to an external math coprocessor. Setting this flag forces all floating-point instructions to be handled by software emulation. Table 12-3 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the EM, MP, and TS flags.

Also, when the EM flag is set, execution of an MMX instruction causes an invalid-opcode exception (#UD) to be generated (see Table 15-1). Thus, if an IA-32 or Intel 64 processor incorporates MMX technology, the EM flag must be set to 0 to enable execution of MMX instructions.

Similarly for SSE/SSE2/SSE3/SSSE3/SSE4 extensions, when the EM flag is set, execution of most SSE/SSE2/SSE3/SSSE3/SSE4 instructions causes an invalid opcode exception (#UD) to be generated (see

Table 16-1). If an IA-32 or Intel 64 processor incorporates the SSE/SSE2/SSE3/SSSE3/SSE4 extensions, the EM flag must be set to 0 to enable execution of these extensions. SSE/SSE2/SSE3/SSSE3/SSE4 instructions not affected by the EM flag include: PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

CR0.MP

Monitor Coprocessor (bit 1 of CR0) — Controls the interaction of the WAIT (or FWAIT) instruction with the TS flag (bit 3 of CR0). If the MP flag is set, a WAIT instruction generates a device-not-available exception (#NM) if the TS flag is also set. If the MP flag is clear, the WAIT instruction ignores the setting of the TS flag. Table 12-3 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the MP, EM, and TS flags.

CR0.PE

Protection Enable (bit 0 of CR0) — Enables protected mode when set; enables real-address mode when clear. This flag does not enable paging directly. It only enables segment-level protection. To enable paging, both the PE and PG flags must be set.

See also: Section 12.9, “Mode Switching.”

CR3.LAM_U48

User LAM48 enable (bit 62 of CR3) — When set and CR3.LAM_U57 is clear, enables LAM48 (masking of linear-address bits 62:48) for user pointers. See Section 4.4, “Linear-Address Masking.”

CR3.LAM_U57

User LAM57 enable (bit 61 of CR3) — When set, enables LAM57 (masking of linear-address bits 62:57) for user pointers and overrides CR3.LAM_U48. See Section 4.4, “Linear-Address Masking.”

CR3.PCD

Page-level Cache Disable (bit 4 of CR3) — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 5.9, “Paging and Memory Typing.” This bit is not used if paging is disabled, with PAE paging, or with 4-level paging¹ or 5-level paging if CR4.PCIDE=1.

CR3.PWT

Page-level Write-Through (bit 3 of CR3) — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 5.9, “Paging and Memory Typing.” This bit is not used if paging is disabled, with PAE paging, or with 4-level paging or 5-level paging if CR4.PCIDE=1.

CR4.VME

Virtual-8086 Mode Extensions (bit 0 of CR4) — Enables interrupt- and exception-handling extensions in virtual-8086 mode when set; disables the extensions when clear. Use of the virtual mode extensions can improve the performance of virtual-8086 applications by eliminating the overhead of calling the virtual-8086 monitor to handle interrupts and exceptions that occur while executing an 8086 program and, instead, redirecting the interrupts and exceptions back to the 8086 program’s handlers. It also provides hardware support for a virtual interrupt flag (VIF) to improve reliability of running 8086 programs in multi-tasking and multiple-processor environments.

See also: Section 23.3, “Interrupt and Exception Handling in Virtual-8086 Mode.”

CR4.FRED

FRED enable (bit 32 of CR4) — When set, enables FRED transitions. See Chapter 8, “Flexible Return and Event Delivery (FRED).”

CR4.LAM_SUP

Supervisor LAM enable (bit 28 of CR4) — When set, enables LAM (linear-address masking) for supervisor pointers. See Section 4.4, “Linear-Address Masking.”

CR4.LASS

Linear-address-space Separation (bit 27 of CR4) — When set, enables LASS (linear-address-space separation). See Section 4.3, “Linear-Address-Space Separation (LASS).”

CR4.UINTR

User Interrupts Enable Bit (bit 25 of CR4) — Enables user interrupts when set, including user-interrupt delivery, user-interrupt notification identification, and the user-interrupt instructions.

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

CR4.PKS

Enable protection keys for supervisor-mode pages (bit 24 of CR4) — 4-level paging and 5-level paging associate each supervisor-mode linear address with a protection key. When set, this flag allows use of the IA32_PKRS MSR to specify, for each protection key, whether supervisor-mode linear addresses with that protection key can be read or written.

CR4.CET

Control-flow Enforcement Technology (bit 23 of CR4) — Enables control-flow enforcement technology when set. See Chapter 18, “Control-flow Enforcement Technology (CET),” of the *IA-32 Intel® Architecture Software Developer’s Manual, Volume 1*. This flag can be set only if CR0.WP is set, and it must be clear before CR0.WP can be cleared (see below).

CR4.PKE

Enable protection keys for user-mode pages (bit 22 of CR4) — 4-level paging and 5-level paging associate each user-mode linear address with a protection key. When set, this flag indicates (via CPUID.07H:ECX.OSPKE[4]) that the operating system supports use of the PKRU register to specify, for each protection key, whether user-mode linear addresses with that protection key can be read or written. This bit also enables access to the PKRU register using the RDPKRU and WRPKRU instructions.

CR4.SMAP

SMAP-Enable Bit (bit 21 of CR4) — Enables supervisor-mode access prevention (SMAP) when set. See Section 5.6, “Access Rights.”

CR4.SMEP

SMEP-Enable Bit (bit 20 of CR4) — Enables supervisor-mode execution prevention (SMEP) when set. See Section 5.6, “Access Rights.”

CR4.KL

Key-Locker-Enable Bit (bit 19 of CR4) — When set, the `LOADIWKEY` instruction is enabled; in addition, if support for the AES Key Locker instructions has been activated by system firmware, CPUID.19H:EBX.AESKLE[0] is enumerated as 1 and the AES Key Locker instructions are enabled.¹ When clear, CPUID.19H:EBX.AESKLE[0] is enumerated as 0 and execution of any Key Locker instruction causes an invalid-opcode exception (#UD).

CR4.OSXSAVE

XSAVE and Processor Extended States-Enable Bit (bit 18 of CR4) — When set, this flag: (1) indicates (via CPUID.01H:ECX.OSXSAVE[27]) that the operating system supports the use of the `XGETBV`, `XSAVE`, and `XRSTOR` instructions by general software; (2) enables the `XSAVE` and `XRSTOR` instructions to save and restore the x87 FPU state (including MMX registers), the SSE state (XMM registers and MXCSR), along with other processor extended states enabled in XCR0; (3) enables the processor to execute `XGETBV` and `XSETBV` instructions in order to read and write XCR0. See Section 2.6 and Chapter 16, “System Programming for Instruction Set Extensions and Processor Extended States.”

CR4.PCIDE

PCID-Enable Bit (bit 17 of CR4) — Enables process-context identifiers (PCIDs) when set. See Section 5.10.1, “Process-Context Identifiers (PCIDs).” Applies only in IA-32e mode (if IA32_EFER.LMA = 1).

CR4.FSGSBASE

FSGSBASE-Enable Bit (bit 16 of CR4) — Enables the instructions `RDFSBASE`, `RDGSBASE`, `WRFSBASE`, and `WRGSBASE`.

CR4.SMXE

SMX-Enable Bit (bit 14 of CR4) — Enables SMX operation when set. See Chapter 7, “Safer Mode Extensions Reference,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

See also: Chapter 5, “Paging.”

1. Software can check CPUID.19H:EBX.AESKLE[0] after setting CR4.KL to determine whether the AES Key Locker instructions have been enabled. Note that some processors may allow enabling of those instructions without activation by system firmware. Some processors may not support use of the AES Key Locker instructions in system-management mode (SMM). Those processors enumerate CPUID.19H:EBX.AESKLE[0] as 0 in SMM regardless of the setting of CR4.KL.

CR4.VMXE

VMX-Enable Bit (bit 13 of CR4) — Enables VMX operation when set. See Chapter 26, “Introduction to Virtual Machine Extensions.”

CR4.LA57

57-bit linear addresses (bit 12 of CR4) — When set in IA-32e mode, the processor uses 5-level paging to translate 57-bit linear addresses. When clear in IA-32e mode, the processor uses 4-level paging to translate 48-bit linear addresses. This bit cannot be modified in IA-32e mode.

CR4.UIMP

User-Mode Instruction Prevention (bit 11 of CR4) — When set, the following instructions cannot be executed if CPL > 0: SGDT, SIDT, SLDT, SMSW, and STR. An attempt at such execution causes a general-protection exception (#GP).

CR4.OSXMMEXCPT

Operating System Support for Unmasked SIMD Floating-Point Exceptions (bit 10 of CR4) — When set, indicates that the operating system supports the handling of unmasked SIMD floating-point exceptions through an exception handler that is invoked when a SIMD floating-point exception (#XM) is generated. SIMD floating-point exceptions are only generated by SSE/SSE2/SSE3/SSE4.1 SIMD floating-point instructions.

The operating system or executive must explicitly set this flag. If this flag is not set, the processor will generate an invalid opcode exception (#UD) whenever it detects an unmasked SIMD floating-point exception.

CR4.OSFXSR

Operating System Support for FXSAVE and FXRSTOR instructions (bit 9 of CR4) — When set, this flag: (1) indicates to software that the operating system supports the use of the FXSAVE and FXRSTOR instructions, (2) enables the FXSAVE and FXRSTOR instructions to save and restore the contents of the XMM and MXCSR registers along with the contents of the x87 FPU and MMX registers, and (3) enables the processor to execute SSE/SSE2/SSE3/SSSE3/SSE4 instructions, with the exception of the PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

If this flag is clear, the FXSAVE and FXRSTOR instructions will save and restore the contents of the x87 FPU and MMX registers, but they may not save and restore the contents of the XMM and MXCSR registers. Also, the processor will generate an invalid opcode exception (#UD) if it attempts to execute any SSE/SSE2/SSE3 instruction, with the exception of PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. The operating system or executive must explicitly set this flag.

NOTE

CPUID feature flag FXSR indicates availability of the FXSAVE/FXRSTOR instructions. The OSFXSR bit provides operating system software with a means of enabling FXSAVE/FXRSTOR to save/restore the contents of the X87 FPU, XMM, and MXCSR registers. Consequently OSFXSR bit indicates that the operating system provides context switch support for SSE/SSE2/SSE3/SSSE3/SSE4.

CR4.PCE

Performance-Monitoring Counter Enable (bit 8 of CR4) — Enables execution of the RDPMC instruction for programs or procedures running at any protection level when set; RDPMC instruction can be executed only at protection level 0 when clear.

CR4.PGE

Page Global Enable (bit 7 of CR4) — (Introduced in the P6 family processors.) Enables the global page feature when set; disables the global page feature when clear. The global page feature allows frequently used or shared pages to be marked as global to all users (done with the global flag, bit 8, in a page-directory-pointer-table entry, a page-directory entry, or a page-table entry). Global pages are not flushed from the translation-lookaside buffer (TLB) on a task switch or a write to register CR3.

When enabling the global page feature, paging must be enabled (by setting the PG flag in control register CR0) before the PGE flag is set. Reversing this sequence may affect program correctness, and processor performance will be impacted.

See also: Section 5.10, “Caching Translation Information.”

CR4.MCE

Machine-Check Enable (bit 6 of CR4) — Enables the machine-check exception when set; disables the machine-check exception when clear.

See also: Chapter 18, “Machine-Check Architecture.”

CR4.PAE

Physical Address Extension (bit 5 of CR4) — When set, enables paging to produce physical addresses with more than 32 bits. When clear, restricts physical addresses to 32 bits. PAE must be set before entering IA-32e mode.

See also: Chapter 5, “Paging.”

CR4.PSE

Page Size Extensions (bit 4 of CR4) — Enables 4-MByte pages with 32-bit paging when set; restricts 32-bit paging to pages of 4 KBytes when clear.

See also: Section 5.3, “32-Bit Paging.”

CR4.DE

Debugging Extensions (bit 3 of CR4) — References to debug registers DR4 and DR5 cause an undefined opcode (#UD) exception to be generated when set; when clear, processor aliases references to registers DR4 and DR5 for compatibility with software written to run on earlier IA-32 processors.

See also: Section 20.2.2, “Debug Registers DR4 and DR5.”

CR4.TSD

Time Stamp Disable (bit 2 of CR4) — Restricts the execution of the RDTSC instruction to procedures running at privilege level 0 when set; allows RDTSC instruction to be executed at any privilege level when clear. This bit also applies to the RDTSCP instruction if supported (if CPUID.80000001H:EDX[27] = 1).

CR4.PVI

Protected-Mode Virtual Interrupts (bit 1 of CR4) — Enables hardware support for a virtual interrupt flag (VIF) in protected mode when set; disables the VIF flag in protected mode when clear.

See also: Section 23.4, “Protected-Mode Virtual Interrupts.”

2.5.1 CPUID Qualification of Control Register Flags

Not all flags in control register CR4 are implemented on all processors. With the exception of the PCE flag, they can be qualified with the CPUID instruction to determine if they are implemented on the processor before they are used.

The CR8 register is available on processors that support Intel 64 architecture.

2.6 EXTENDED CONTROL REGISTERS (INCLUDING XCR0)

If CPUID.01H:ECX.XSAVE[26] is 1, the processor supports one or more **extended control registers** (XCRs). Currently, the only such register defined is XCR0. This register specifies the set of processor state components for which the operating system provides context management, e.g., x87 FPU state, SSE state, AVX state. The OS programs XCR0 to reflect the features for which it provides context management.

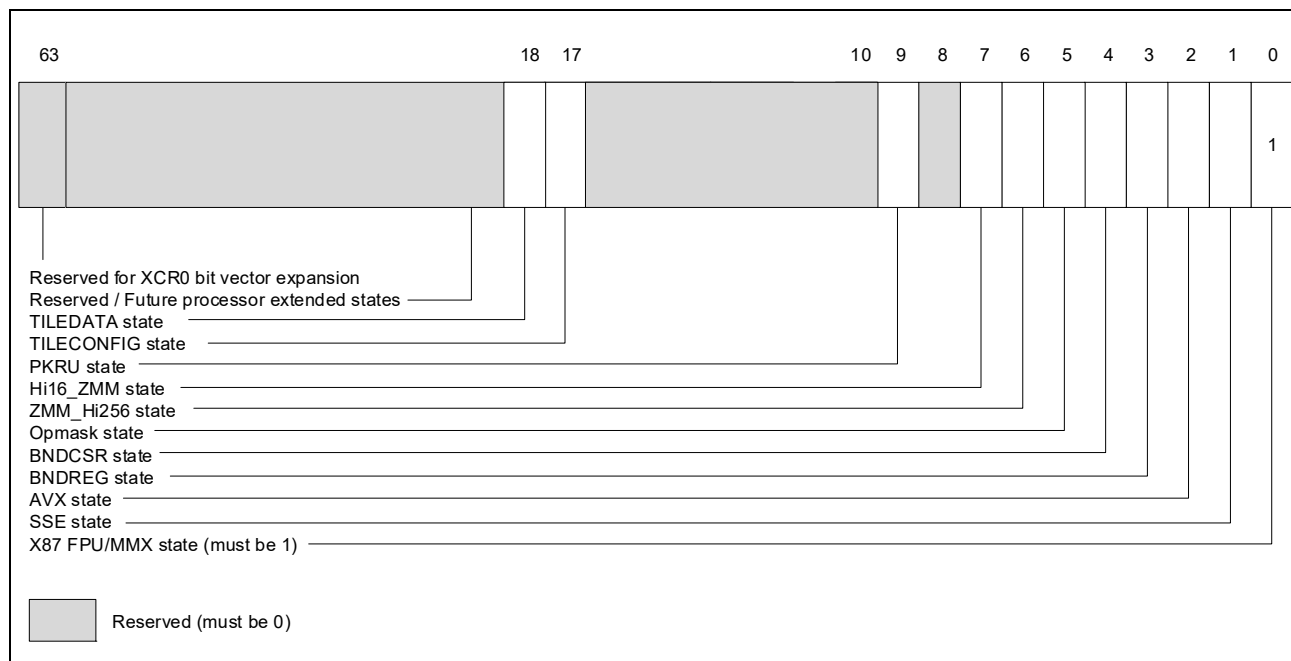


Figure 2-8. XCR0

Software can access XCR0 only if CR4.OSXSAVE[bit 18] = 1. (This bit is also readable as CPUID.01H:ECX.OSXSAVE[27].) Software can use CPUID.0DH to enumerate the bits in XCR0 that the processor supports (see CPUID instruction in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A). Each supported state component is represented by a bit in XCR0. System software enables state components by loading an appropriate bit mask value into XCR0 using the XSETBV instruction.

As each bit in XCR0 (except bit 63) corresponds to a processor state component, XCR0 thus provides support for up to 63 sets of processor state components. Bit 63 of XCR0 is reserved for future expansion and will not represent a processor state component.

Currently, XCR0 defines support for the following state components:

- XCR0.X87 (bit 0): This bit 0 must be 1. An attempt to write 0 to this bit causes a #GP exception.
- XCR0.SSE (bit 1): If 1, the XSAVE feature set can be used to manage MXCSR and the XMM registers (XMM0-XMM15 in 64-bit mode; otherwise XMM0-XMM7).
- XCR0.AVX (bit 2): If 1, Intel AVX instructions can be executed and the XSAVE feature set can be used to manage the upper halves of the YMM registers (YMM0-YMM15 in 64-bit mode; otherwise YMM0-YMM7).
- XCR0.BNDREG (bit 3): If 1, Intel MPX instructions can be executed and the XSAVE feature set can be used to manage the bounds registers BND0-BND3.
- XCR0.BNDCSR (bit 4): If 1, Intel MPX instructions can be executed and the XSAVE feature set can be used to manage the BNDCFGU and BNDSTATUS registers.
- XCR0.opmask (bit 5): If 1, Intel AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the opmask registers k0-k7.
- XCR0.ZMM_Hi256 (bit 6): If 1, Intel AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the upper halves of the lower ZMM registers (ZMM0-ZMM15 in 64-bit mode; otherwise ZMM0-ZMM7).
- XCR0.Hi16_ZMM (bit 7): If 1, Intel AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the upper ZMM registers (ZMM16-ZMM31, only in 64-bit mode).
- XCR0.PKRU (bit 9): If 1, the XSAVE feature set can be used to manage the PKRU register (see Section 2.7).
- XCR0.TILECFG (bit 17): If 1, and if XCR0.TILEDATA is also 1, Intel AMX instructions can be executed and the XSAVE feature set can be used to manage TILECFG.

- XCR0.TILEDATA (bit 18): If 1, and if XCR0.TILECFG is also 1, Intel AMX instructions can be executed and the XSAVE feature set can be used to manage TILEDATA.

An attempt to use XSETBV to write to XCR0 results in general-protection exceptions (#GP) if it would do any of the following:

- Set a bit reserved in XCR0 for a given processor (as determined by the contents of EAX and EDX after executing CPUID.0DH.00H).
- Clear XCR0.x87.
- Clear XCR0.SSE and set XCR0.AVX.
- Clear XCR0.AVX and set any of XCR0.opmask, XCR0.ZMM_Hi256, or XCR0.Hi16_ZMM.
- Set either XCR0.BNDREG or XCR0.BNDCSR while not setting the other.
- Set any of XCR0.opmask, XCR0.ZMM_Hi256, and XCR0.Hi16_ZMM while not setting all of them.
- Set either XCR0.TILECFG or XCR0.TILEDATA while not setting the other.

After reset, all bits (except bit 0) in XCR0 are cleared to zero; XCR0[0] is set to 1.

2.7 PROTECTION-KEY RIGHTS REGISTERS (PKRU AND IA32_PKRS)

Processors may support either or both of two protection-key rights registers: PKRU for user-mode pages and the IA32_PKRS MSR (MSR index 6E1H) for supervisor-mode pages. 4-level paging and 5-level paging associate a 4-bit **protection key** with each page. The protection-key rights registers determine accessibility based on a page's protection key.

If CPUID.07H.00H:ECX.PKU[3] = 1, the processor supports the protection-key feature for user-mode pages. When CR4.PKE = 1, software can use the **protection-key rights register for user pages** (PKRU) to specify the access rights for user-mode pages for each protection key.

If CPUID.07H.00H:ECX.PKS[31] = 1, the processor supports the protection-key feature for supervisor-mode pages. When CR4.PKS = 1, software can use the **protection-key rights register for supervisor pages** (the IA32_PKRS MSR) to specify the access rights for supervisor-mode pages for each protection key.

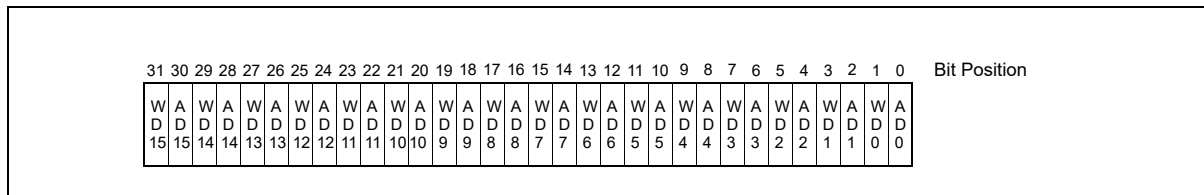


Figure 2-9. Format of Protection-Key Rights Registers

The format of each protection-key rights register is given in Figure 2-9. Each contains 16 pairs of disable controls to prevent data accesses to linear addresses (user-mode or supervisor-mode, depending on the register) based on their protection keys. Each protection key i ($0 \leq i \leq 15$) is associated with two bits in each protection-key rights register:

- Bit $2i$, shown as "AD i " (access disable): if set, the processor prevents any data accesses to linear addresses (user-mode or supervisor-mode, depending on the register) with protection key i .
- Bit $2i+1$, shown as "WD i " (write disable): if set, the processor prevents write accesses to linear addresses (user-mode or supervisor-mode, depending on the register) with protection key i .

(Bits 63:32 of the IA32_PKRS MSR are reserved and must be zero.)

See Section 5.6.2, "Protection Keys," for details of how the processor uses the protection-key rights registers to control accesses to linear addresses.

Software can read and write PKRU using the RDPKRU and WRPKRU instructions. The IA32_PKRS MSR can be read and written with the RDMSR and WRMSR instructions. Writes to the IA32_PKRS MSR using WRMSR are not serializing.

2.8 SYSTEM INSTRUCTION SUMMARY

System instructions handle system-level functions such as loading system registers, managing the cache, managing interrupts, or setting up the debug registers. Many of these instructions can be executed only by operating-system or executive procedures (that is, procedures running at privilege level 0). Others can be executed at any privilege level and are thus available to application programs.

Table 2-3 lists the system instructions and indicates whether they are available and useful for application programs. These instructions are described in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D.

Table 2-3. Summary of System Instructions

Instruction	Description	Useful to Application?	Protected from Application?
LLDT	Load LDT Register	No	Yes
SLDT	Store LDT Register	No	If CR4.UMIP = 1
LGDT	Load GDT Register	No	Yes
SGDT	Store GDT Register	No	If CR4.UMIP = 1
LTR	Load Task Register	No	Yes
STR	Store Task Register	No	If CR4.UMIP = 1
LIDT	Load IDT Register	No	Yes
SIDT	Store IDT Register	No	If CR4.UMIP = 1
MOV CR _n	Load and store control registers	No	Yes
SMSW	Store MSW	Yes	If CR4.UMIP = 1
LMSW	Load MSW	No	Yes
CLTS	Clear TS flag in CRO	No	Yes
ARPL	Adjust RPL	Yes ^{1, 5}	No
LAR	Load Access Rights	Yes	No
LSL	Load Segment Limit	Yes	No
VERR	Verify for Reading	Yes	No
VERW	Verify for Writing	Yes	No
MOV DR _n	Load and store debug registers	No	Yes
INVD	Invalidate cache, no writeback	No	Yes
WBINVD	Invalidate cache, with writeback	No	Yes
INVLPG	Invalidate TLB entry	No	Yes
HLT	Halt Processor	No	Yes
LOCK (Prefix)	Bus Lock	Yes	No
RSM	Return from system management mode	No	Yes
RDMSR ³ , RDMSRLIST	Read Model-Specific Registers	No	Yes
WRMSR ³ , WRMSRLIST, WRMSRNS	Write Model-Specific Registers	No	Yes
RDPMS ⁴	Read Performance-Monitoring Counter	Yes	Yes ²
RDTSC ³	Read Time-Stamp Counter	Yes	Yes ²
RDTSCP ⁷	Read Serialized Time-Stamp Counter	Yes	Yes ²
SWAPGS	Swap GS Base Register	No	Yes

Table 2-3. Summary of System Instructions (Contd.)

Instruction	Description	Useful to Application?	Protected from Application?
XGETBV	Return the state of XCRO	Yes	No
XSETBV	Enable one or more processor extended states	No ⁶	Yes
LKGS	Load Kernel GS Base	No	Yes
IRET	Interrupt Return	Yes	No
SYSEXIT	Fast Return from Fast System Call	No	Yes
SYSRET	Return from Fast System Call	No	Yes
ERETS	Event Return to Supervisor	No	Yes
ERETU	Event Return to User	No	Yes

NOTES:

1. Useful to application programs running at a CPL of 1 or 2.
2. The TSD and PCE flags in control register CR4 control access to these instructions by application programs running at a CPL of 3.
3. These instructions were introduced into the IA-32 Architecture with the Pentium processor.
4. This instruction was introduced into the IA-32 Architecture with the Pentium Pro processor and the Pentium processor with MMX technology.
5. This instruction is not supported in 64-bit mode.
6. Application uses XGETBV to query which set of processor extended states are enabled.
7. RDTSCP is introduced in Intel Core i7 processor.

2.8.1 Loading and Storing System Registers

The GDTR, LDTR, IDTR, and TR registers each have a load and store instruction for loading data into and storing data from the register:

- **LGDT (Load GDTR Register)** — Loads the GDT base address and limit from memory into the GDTR register.
- **SGDT (Store GDTR Register)** — Stores the GDT base address and limit from the GDTR register into memory.
- **LIDT (Load IDTR Register)** — Loads the IDT base address and limit from memory into the IDTR register.
- **SIDT (Store IDTR Register)** — Stores the IDT base address and limit from the IDTR register into memory.
- **LLDT (Load LDTR Register)** — Loads the LDT segment selector and segment descriptor from memory into the LDTR. (The segment selector operand can also be located in a general-purpose register.)
- **SLDT (Store LDTR Register)** — Stores the LDT segment selector from the LDTR register into memory or a general-purpose register.
- **LTR (Load Task Register)** — Loads segment selector and segment descriptor for a TSS from memory into the task register. (The segment selector operand can also be located in a general-purpose register.)
- **STR (Store Task Register)** — Stores the segment selector for the current task TSS from the task register into memory or a general-purpose register.

The LMSW (load machine status word) and SMSW (store machine status word) instructions operate on bits 0 through 15 of control register CR0. These instructions are provided for compatibility with the 16-bit Intel 286 processor. Programs written to run on 32-bit IA-32 processors should not use these instructions. Instead, they should access the control register CR0 using the MOV CR instruction.

The CLTS (clear TS flag in CR0) instruction is provided for use in handling a device-not-available exception (#NM) that occurs when the processor attempts to execute a floating-point instruction when the TS flag is set. This instruction allows the TS flag to be cleared after the x87 FPU context has been saved, preventing further #NM exceptions. See Section 2.5, “Control Registers,” for more information on the TS flag.

The control registers (CR0, CR1, CR2, CR3, CR4, and CR8) are loaded using the MOV instruction. The instruction loads a control register from a general-purpose register or stores the content of a control register in a general-purpose register.

2.8.2 Verifying of Access Privileges

The processor provides several instructions for examining segment selectors and segment descriptors to determine if access to their associated segments is allowed. These instructions duplicate some of the automatic access rights and type checking done by the processor, thus allowing operating-system or executive software to prevent exceptions from being generated.

The ARPL (adjust RPL) instruction adjusts the RPL (requestor privilege level) of a segment selector to match that of the program or procedure that supplied the segment selector. See Section 6.10.4, “Checking Caller Access Privileges (ARPL Instruction),” for a detailed explanation of the function and use of this instruction. Note that ARPL is not supported in 64-bit mode.

The LAR (load access rights) instruction verifies the accessibility of a specified segment and loads access rights information from the segment’s segment descriptor into a general-purpose register. Software can then examine the access rights to determine if the segment type is compatible with its intended use. See Section 6.10.1, “Checking Access Rights (LAR Instruction),” for a detailed explanation of the function and use of this instruction.

The LSL (load segment limit) instruction verifies the accessibility of a specified segment and loads the segment limit from the segment’s segment descriptor into a general-purpose register. Software can then compare the segment limit with an offset into the segment to determine whether the offset lies within the segment. See Section 6.10.3, “Checking That the Pointer Offset Is Within Limits (LSL Instruction),” for a detailed explanation of the function and use of this instruction.

The VERR (verify for reading) and VERW (verify for writing) instructions verify if a selected segment is readable or writable, respectively, at a given CPL. See Section 6.10.2, “Checking Read/Write Rights (VERR and VERW Instructions),” for a detailed explanation of the function and use of these instructions.

2.8.3 Loading and Storing Debug Registers

Internal debugging facilities in the processor are controlled by a set of 8 debug registers (DR0-DR7). The MOV instruction allows setup data to be loaded to and stored from these registers.

On processors that support Intel 64 architecture, debug registers DR0-DR7 are 64 bits. In 32-bit modes and compatibility mode, writes to a debug register fill the upper 32 bits with zeros. Reads return the lower 32 bits. In 64-bit mode, the upper 32 bits of DR6-DR7 are reserved and must be written with zeros. Writing one to any of the upper 32 bits causes an exception, #GP(0).

In 64-bit mode, MOV DRn instructions read or write all 64 bits of a debug register (operand-size prefixes are ignored). All 64 bits of DR0-DR3 are writable by software. However, MOV DRn instructions do not check that addresses written to DR0-DR3 are in the limits of the implementation. Address matching is supported only on valid addresses generated by the processor implementation.

2.8.4 Invalidating Caches and TLBs

The processor provides several instructions for use in explicitly invalidating its caches and TLB entries. The INVD (invalidate cache with no writeback) instruction invalidates all data and instruction entries in the internal caches and sends a signal to the external caches indicating that they should also be invalidated.

The WBINVD (invalidate cache with writeback) instruction performs the same function as the INVD instruction, except that it writes back modified lines in its internal caches to memory before it invalidates the caches. After invalidating the caches local to the executing logical processor or processor core, WBINVD signals caches higher in the cache hierarchy (caches shared with the invalidating logical processor or core) to write back any data they have in modified state at the time of instruction execution and to invalidate their contents.

Note, non-shared caches may not be written back nor invalidated. In Figure 2-10 below, if code executing on either LP0 or LP1 were to execute a WBINVD, the shared L1 and L2 for LP0/LP1 will be written back and invalidated as will the shared L3. However, the L1 and L2 caches not shared with LP0 and LP1 will not be written back nor invalidated.

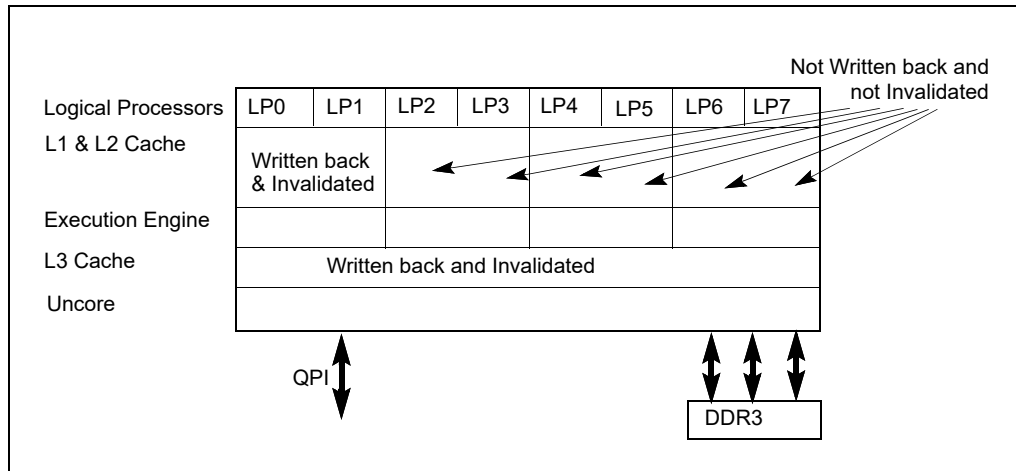


Figure 2-10. WBINVD Invalidation of Shared and Non-Shared Cache Hierarchy

The INVLPG (invalidate TLB entry) instruction invalidates (flushes) the TLB entry for a specified page.

2.8.5 Controlling the Processor

The HLT (halt processor) instruction stops the processor until an enabled interrupt (such as NMI or SMI, which are normally enabled), a debug exception, the BINIT# signal, the INIT# signal, or the RESET# signal is received. The processor generates a special bus cycle to indicate that the halt mode has been entered.

Hardware may respond to this signal in a number of ways. An indicator light on the front panel may be turned on. An NMI interrupt for recording diagnostic information may be generated. Reset initialization may be invoked (note that the BINIT# pin was introduced with the Pentium Pro processor). If any non-wake events are pending during shutdown, they will be handled after the wake event from shutdown is processed (for example, A20M# interrupts).

The LOCK prefix invokes a locked (atomic) read-modify-write operation when modifying a memory operand. This mechanism is used to allow reliable communications between processors in multiprocessor systems, as described below:

- In the Pentium processor and earlier IA-32 processors, the LOCK prefix causes the processor to assert the LOCK# signal during the instruction. This always causes an explicit bus lock to occur.
- In the Pentium 4, Intel Xeon, and P6 family processors, the locking operation is handled with either a cache lock or bus lock. If a memory access is cacheable and affects only a single cache line, a cache lock is invoked and the system bus and the actual memory location in system memory are not locked during the operation. Here, other Pentium 4, Intel Xeon, or P6 family processors on the bus write-back any modified data and invalidate their caches as necessary to maintain system memory coherency. If the memory access is not cacheable and/or it crosses a cache line boundary, the processor's LOCK# signal is asserted and the processor does not respond to requests for bus control during the locked operation.

The RSM (return from SMM) instruction restores the processor (from a context dump) to the state it was in prior to a system management mode (SMM) interrupt.

2.8.6 Reading Performance-Monitoring and Time-Stamp Counters

The RDPMC (read performance-monitoring counter) and RDTSC (read time-stamp counter) instructions allow application programs to read the processor's performance-monitoring and time-stamp counters, respectively. Processors based on Intel NetBurst® microarchitecture have eighteen 40-bit performance-monitoring counters; P6 family processors have two 40-bit counters. Intel Atom® processors and most of the processors based on the Intel Core microarchitecture support two types of performance monitoring counters: programmable performance counters similar to those available in the P6 family, and three fixed-function performance monitoring counters.

Details of programmable and fixed-function performance monitoring counters for each processor generation are described in Chapter 21, “Last Branch Records.”

The programmable performance counters can support counting either the occurrence or duration of events. Events that can be monitored on programmable counters generally are model specific (except for architectural performance events enumerated by CPUID.0AH); they may include the number of instructions decoded, interrupts received, or the number of cache loads. Individual counters can be set up to monitor different events. Use the system instruction WRMSR to set up values in one of the IA32_PERFECTSELx MSR, in one of the 45 ESCRs and one of the 18 CCCR MSRs (for Pentium 4 and Intel Xeon processors); or in the PerfEvtSel0 or the PerfEvtSel1 MSR (for the P6 family processors). The RDPMC instruction loads the current count from the selected counter into the EDX:EAX registers.

Fixed-function performance counters record only specific events that are defined at: <https://perfmon-events.intel.com/>, and the width/number of fixed-function counters are enumerated by CPUID.0AH.

The time-stamp counter is a model-specific 64-bit counter that is reset to zero each time the processor is reset. If not reset, the counter will increment $\sim 9.5 \times 10^{16}$ times per year when the processor is operating at a clock rate of 3GHz. At this clock frequency, it would take over 190 years for the counter to wrap around. The RDTSC instruction loads the current count of the time-stamp counter into the EDX:EAX registers.

See Section 22.1, “Performance Monitoring Overview,” and Section 20.17, “Time-Stamp Counter,” for more information about the performance monitoring and time-stamp counters.

The RDTSC instruction was introduced into the IA-32 architecture with the Pentium processor. The RDPMC instruction was introduced into the IA-32 architecture with the Pentium Pro processor and the Pentium processor with MMX technology. Earlier Pentium processors have two performance-monitoring counters, but they can be read only with the RDMSR instruction, and only at privilege level 0.

2.8.6.1 Reading Counters in 64-Bit Mode

In 64-bit mode, RDTSC operates the same as in protected mode. The count in the time-stamp counter is stored in EDX:EAX (or RDX[31:0]:RAX[31:0] with RDX[63:32]:RAX[63:32] cleared).

RDPMC requires an index to specify the offset of the performance-monitoring counter. In 64-bit mode for Pentium 4 or Intel Xeon processor families, the index is specified in ECX[30:0]. The current count of the performance-monitoring counter is stored in EDX:EAX (or RDX[31:0]:RAX[31:0] with RDX[63:32]:RAX[63:32] cleared).

2.8.7 Reading and Writing Model-Specific Registers

The RDMSR (read model-specific register) and WRMSR (write model-specific register) instructions allow a processor's 64-bit model-specific registers (MSRs) to be read and written, respectively. The MSR to be read or written is specified by the value in the ECX register.

RDMSR reads the value from the specified MSR to the EDX:EAX registers; WRMSR writes the value in the EDX:EAX registers to the specified MSR. RDMSR and WRMSR were introduced into the IA-32 architecture with the Pentium processor.

See Section 12.4, “Model-Specific Registers (MSRs),” for more information.

2.8.7.1 Reading and Writing Model-Specific Registers in 64-Bit Mode

RDMSR and WRMSR require an index to specify the address of an MSR. In 64-bit mode, the index is 32 bits; it is specified using ECX.

2.8.8 Enabling Processor Extended States

The XSETBV instruction is required to enable OS support of individual processor extended states in XCR0 (see Section 2.6).

12. Updates to Chapter 6, Volume 3A

Change bars and text show changes to Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- Documented that call gates cannot be used when FRED transitions are enabled in Section 6.8.3.1, "IA-32e Mode Call Gates".
- In Section 6.8.8, "Fast System Calls in 64-Bit Mode", documented when SYSCALL, SYSENTER, and ERETU (or ERETS) are used for FRED transitions.
- In Section 6.9, "Privileged Instructions", added RDMSRLIST, WRMSRLIST, WRMSRNS, ERETS, ERETU, and LKGS.

In protected mode, the Intel 64 and IA-32 architectures provide a protection mechanism that operates at both the segment level and the page level. This protection mechanism provides the ability to limit access to certain segments or pages based on privilege levels (four privilege levels for segments and two privilege levels for pages). For example, critical operating-system code and data can be protected by placing them in more privileged segments than those that contain applications code. The processor's protection mechanism will then prevent application code from accessing the operating-system code and data in any but a controlled, defined manner.

Segment and page protection can be used at all stages of software development to assist in localizing and detecting design problems and bugs. It can also be incorporated into end-products to offer added robustness to operating systems, utilities software, and applications software.

When the protection mechanism is used, each memory reference is checked to verify that it satisfies various protection checks. All checks are made before the memory cycle is started; any violation results in an exception. Because checks are performed in parallel with address translation, there is no performance penalty. The protection checks that are performed fall into the following categories:

- Limit checks.
- Type checks.
- Privilege level checks.
- Restriction of addressable domain.
- Restriction of procedure entry-points.
- Restriction of instruction set.

All protection violation results in an exception being generated. See Chapter 7, "Interrupt and Exception Handling," for an explanation of the exception mechanism. This chapter describes the protection mechanism and the violations which lead to exceptions.

The following sections describe the protection mechanism available in protected mode. See Chapter 23, "8086 Emulation," for information on protection in real-address and virtual-8086 mode.

6.1 ENABLING AND DISABLING SEGMENT AND PAGE PROTECTION

Setting the PE flag in register CR0 causes the processor to switch to protected mode, which in turn enables the segment-protection mechanism. Once in protected mode, there is no control bit for turning the protection mechanism on or off. The part of the segment-protection mechanism that is based on privilege levels can essentially be disabled while still in protected mode by assigning a privilege level of 0 (most privileged) to all segment selectors and segment descriptors. This action disables the privilege level protection barriers between segments, but other protection checks such as limit checking and type checking are still carried out.

Page-level protection is automatically enabled when paging is enabled (by setting the PG flag in register CR0). Here again there is no mode bit for turning off page-level protection once paging is enabled. However, page-level protection can be disabled by performing the following operations:

- Clear the WP flag in control register CR0.
- Set the read/write (R/W) and user/supervisor (U/S) flags for each page-directory and page-table entry.

This action makes each page a writable, user page, which in effect disables page-level protection.

6.2 FIELDS AND FLAGS USED FOR SEGMENT-LEVEL AND PAGE-LEVEL PROTECTION

The processor's protection mechanism uses the following fields and flags in the system data structures to control access to segments and pages:

- **Descriptor type (S) flag** — (Bit 12 in the second doubleword of a segment descriptor.) Determines if the segment descriptor is for a system segment or a code or data segment.
- **Type field** — (Bits 8 through 11 in the second doubleword of a segment descriptor.) Determines the type of code, data, or system segment.
- **Limit field** — (Bits 0 through 15 of the first doubleword and bits 16 through 19 of the second doubleword of a segment descriptor.) Determines the size of the segment, along with the G flag and E flag (for data segments).
- **G flag** — (Bit 23 in the second doubleword of a segment descriptor.) Determines the size of the segment, along with the limit field and E flag (for data segments).
- **E flag** — (Bit 10 in the second doubleword of a data-segment descriptor.) Determines the size of the segment, along with the limit field and G flag.
- **Descriptor privilege level (DPL) field** — (Bits 13 and 14 in the second doubleword of a segment descriptor.) Determines the privilege level of the segment.
- **Requested privilege level (RPL) field** — (Bits 0 and 1 of any segment selector.) Specifies the requested privilege level of a segment selector.
- **Current privilege level (CPL) field** — (Bits 0 and 1 of the CS segment register.) Indicates the privilege level of the currently executing program or procedure. The term current privilege level (CPL) refers to the setting of this field.
- **User/supervisor (U/S) flag** — (Bit 2 of paging-structure entries.) Determines the type of page: user or supervisor.
- **Read/write (R/W) flag** — (Bit 1 of paging-structure entries.) Determines the type of access allowed to a page: read-only or read/write.
- **Execute-disable (XD) flag** — (Bit 63 of certain paging-structure entries.) Determines the type of access allowed to a page: executable or not-executable.

Figure 6-1 shows the location of the various fields and flags in the data-, code-, and system-segment descriptors; Figure 3-6 shows the location of the RPL (or CPL) field in a segment selector (or the CS register); and Chapter 5 identifies the locations of the U/S, R/W, and XD flags in the paging-structure entries.

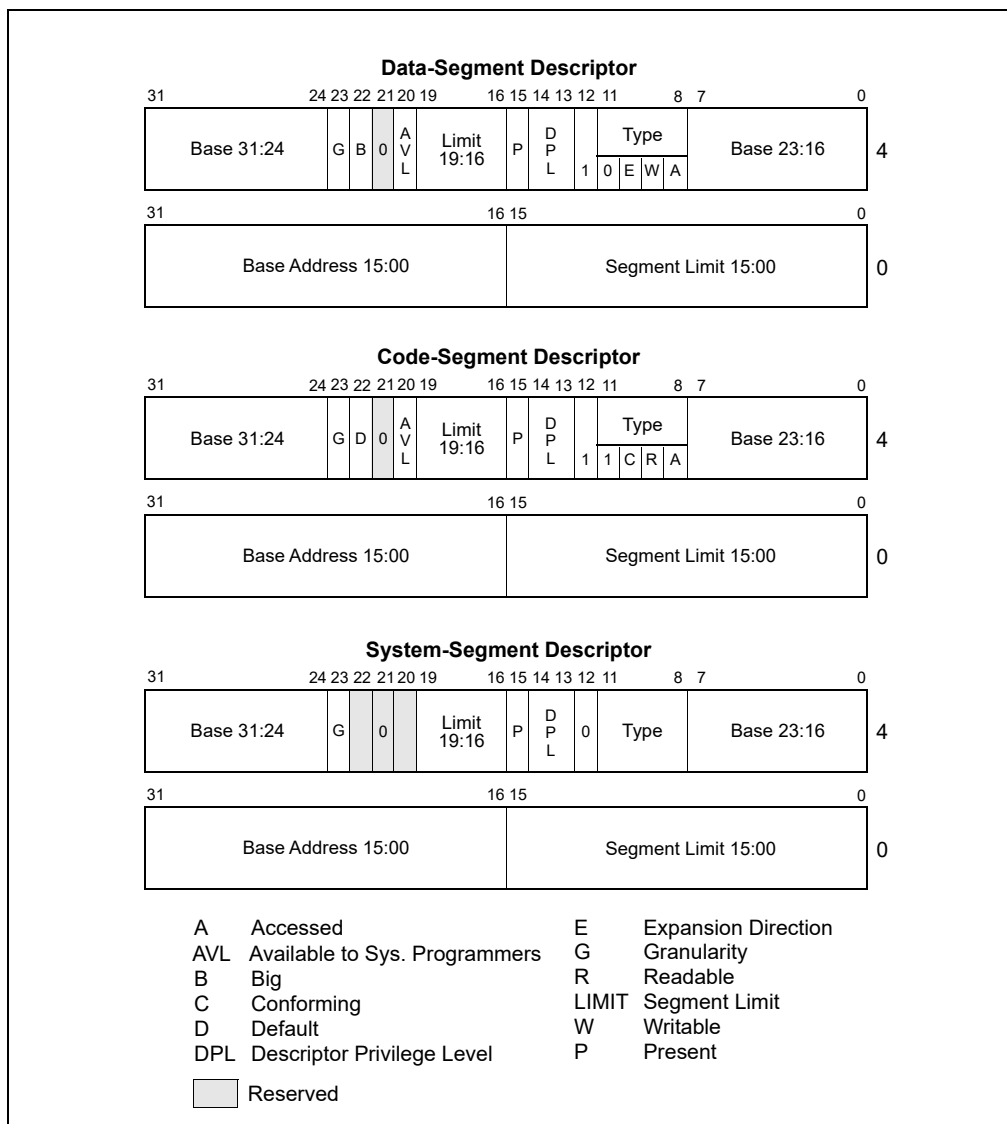


Figure 6-1. Descriptor Fields Used for Protection

Many different styles of protection schemes can be implemented with these fields and flags. When the operating system creates a descriptor, it places values in these fields and flags in keeping with the particular protection style chosen for an operating system or executive. Application programs do not generally access or modify these fields and flags.

The following sections describe how the processor uses these fields and flags to perform the various categories of checks described in the introduction to this chapter.

6.2.1 Code-Segment Descriptor in 64-bit Mode

Code segments continue to exist in 64-bit mode even though, for address calculations, the segment base is treated as zero. Some code-segment (CS) descriptor content (the base address and limit fields) is ignored; the remaining fields function normally (except for the readable bit in the type field).

Code segment descriptors and selectors are needed in IA-32e mode to establish the processor's operating mode and execution privilege-level. The usage is as follows:

- IA-32e mode uses a previously unused bit in the CS descriptor. Bit 53 is defined as the 64-bit (L) flag and is used to select between 64-bit mode and compatibility mode when IA-32e mode is active (IA32_EFER.LMA = 1). See Figure 6-2.
 - If CS.L = 0 and IA-32e mode is active, the processor is running in compatibility mode. In this case, CS.D selects the default size for data and addresses. If CS.D = 0, the default data and address size is 16 bits. If CS.D = 1, the default data and address size is 32 bits.
 - If CS.L = 1 and IA-32e mode is active, the only valid setting is CS.D = 0. This setting indicates a default operand size of 32 bits and a default address size of 64 bits. The CS.L = 1 and CS.D = 1 bit combination is reserved for future use and a #GP fault will be generated on an attempt to use a code segment with these bits set in IA-32e mode.
- In IA-32e mode, the CS descriptor’s DPL is used for execution privilege checks (as in legacy 32-bit mode).

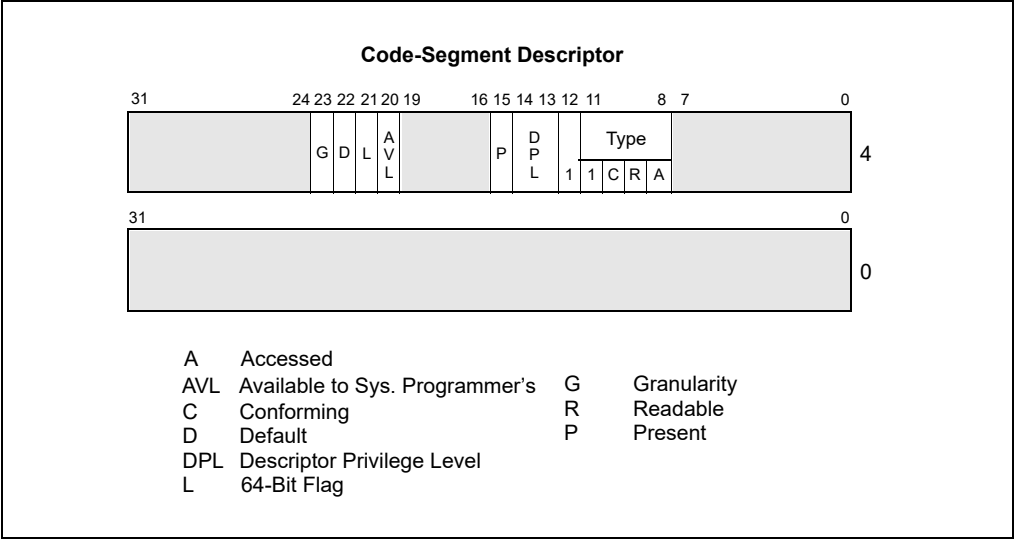


Figure 6-2. Descriptor Fields with Flags used in IA-32e Mode

6.3 LIMIT CHECKING

The limit field of a segment descriptor prevents programs or procedures from addressing memory locations outside the segment. The effective value of the limit depends on the setting of the G (granularity) flag (see Figure 6-1). For data segments, the limit also depends on the E (expansion direction) flag and the B (default stack pointer size and/or upper bound) flag. The E flag is one of the bits in the type field when the segment descriptor is for a data-segment type.

When the G flag is clear (byte granularity), the effective limit is the value of the 20-bit limit field in the segment descriptor. Here, the limit ranges from 0 to FFFFFH (1 MByte). When the G flag is set (4-KByte page granularity), the processor scales the value in the limit field by a factor of 2^{12} (4 KBytes). In this case, the effective limit ranges from FFFH (4 KBytes) to FFFFFFFFH (4 GBytes). Note that when scaling is used (G flag is set), the lower 12 bits of a segment offset (address) are not checked against the limit; for example, note that if the segment limit is 0, offsets 0 through FFFH are still valid.

For all types of segments except expand-down data segments, the effective limit is the last address that is allowed to be accessed in the segment, which is one less than the size, in bytes, of the segment. The processor causes a general-protection exception (or, if the segment is SS, a stack-fault exception) any time an attempt is made to access the following addresses in a segment:

- A byte at an offset greater than the effective limit
- A word at an offset greater than the (effective-limit – 1)
- A doubleword at an offset greater than the (effective-limit – 3)
- A quadword at an offset greater than the (effective-limit – 7)

- A double quadword at an offset greater than the (effective limit – 15)

When the effective limit is FFFFFFFFH (4 GBytes), these accesses may or may not cause the indicated exceptions. Behavior is implementation-specific and may vary from one execution to another.

For expand-down data segments, the segment limit has the same function but is interpreted differently. Here, the effective limit specifies the last address that is not allowed to be accessed within the segment; the range of valid offsets is from (effective-limit + 1) to FFFFFFFFH if the B flag is set and from (effective-limit + 1) to FFFFH if the B flag is clear. An expand-down segment has maximum size when the segment limit is 0.

Limit checking catches programming errors such as runaway code, runaway subscripts, and invalid pointer calculations. These errors are detected when they occur, so identification of the cause is easier. Without limit checking, these errors could overwrite code or data in another segment.

In addition to checking segment limits, the processor also checks descriptor table limits. The GDTR and IDTR registers contain 16-bit limit values that the processor uses to prevent programs from selecting a segment descriptors outside the respective descriptor tables. The LDTR and task registers contain 32-bit segment limit value (read from the segment descriptors for the current LDT and TSS, respectively). The processor uses these segment limits to prevent accesses beyond the bounds of the current LDT and TSS. See Section 3.5.1, “Segment Descriptor Tables,” for more information on the GDT and LDT limit fields; see Section 7.10, “Interrupt Descriptor Table (IDT),” for more information on the IDT limit field; and see Section 10.2.4, “Task Register,” for more information on the TSS segment limit field.

6.3.1 Limit Checking in 64-bit Mode

In 64-bit mode, the processor does not perform runtime limit checking on code or data segments. However, the processor does check descriptor-table limits.

6.4 TYPE CHECKING

Segment descriptors contain type information in two places:

- The S (descriptor type) flag.
- The type field.

The processor uses this information to detect programming errors that result in an attempt to use a segment or gate in an incorrect or unintended manner.

The S flag indicates whether a descriptor is a system type or a code or data type. The type field provides 4 additional bits for use in defining various types of code, data, and system descriptors. Table 3-1 shows the encoding of the type field for code and data descriptors; Table 3-2 shows the encoding of the field for system descriptors.

The processor examines type information at various times while operating on segment selectors and segment descriptors. The following list gives examples of typical operations where type checking is performed (this list is not exhaustive):

- **When a segment selector is loaded into a segment register** — Certain segment registers can contain only certain descriptor types, for example:
 - The CS register only can be loaded with a selector for a code segment.
 - Segment selectors for code segments that are not readable or for system segments cannot be loaded into data-segment registers (DS, ES, FS, and GS).
 - Only segment selectors of writable data segments can be loaded into the SS register.
- When a segment selector is loaded into the LDTR or task register — For example:
 - The LDTR can only be loaded with a selector for an LDT.
 - The task register can only be loaded with a segment selector for a TSS.
- **When instructions access segments whose descriptors are already loaded into segment registers** — Certain segments can be used by instructions only in certain predefined ways, for example:
 - No instruction may write into an executable segment.

- No instruction may write into a data segment if it is not writable.
- No instruction may read an executable segment unless the readable flag is set.
- **When an instruction operand contains a segment selector** — Certain instructions can access segments or gates of only a particular type, for example:
 - A far CALL or far JMP instruction can only access a segment descriptor for a conforming code segment, nonconforming code segment, call gate, task gate, or TSS.
 - The LLDT instruction must reference a segment descriptor for an LDT.
 - The LTR instruction must reference a segment descriptor for a TSS.
 - The LAR instruction must reference a segment or gate descriptor for an LDT, TSS, call gate, task gate, code segment, or data segment.
 - The LSL instruction must reference a segment descriptor for a LDT, TSS, code segment, or data segment.
 - IDT entries must be interrupt, trap, or task gates.
- **During certain internal operations** — For example:
 - On a far call or far jump (executed with a far CALL or far JMP instruction), the processor determines the type of control transfer to be carried out (call or jump to another code segment, a call or jump through a gate, or a task switch) by checking the type field in the segment (or gate) descriptor pointed to by the segment (or gate) selector given as an operand in the CALL or JMP instruction. If the descriptor type is for a code segment or call gate, a call or jump to another code segment is indicated; if the descriptor type is for a TSS or task gate, a task switch is indicated.
 - On a call or jump through a call gate (or on an interrupt- or exception-handler call through a trap or interrupt gate), the processor automatically checks that the segment descriptor being pointed to by the gate is for a code segment.
 - On a call or jump to a new task through a task gate (or on an interrupt- or exception-handler call to a new task through a task gate), the processor automatically checks that the segment descriptor being pointed to by the task gate is for a TSS.
 - On a call or jump to a new task by a direct reference to a TSS, the processor automatically checks that the segment descriptor being pointed to by the CALL or JMP instruction is for a TSS.
 - On return from a nested task (initiated by an IRET instruction), the processor checks that the previous task link field in the current TSS points to a TSS.

6.4.1 Null Segment Selector Checking

Attempting to load a null segment selector (see Section 3.4.2, “Segment Selectors”) into the CS or SS segment register generates a general-protection exception (#GP). A null segment selector can be loaded into the DS, ES, FS, or GS register, but any attempt to access a segment through one of these registers when it is loaded with a null segment selector results in a #GP exception being generated. Loading unused data-segment registers with a null segment selector is a useful method of detecting accesses to unused segment registers and/or preventing unwanted accesses to data segments.

6.4.1.1 NULL Segment Checking in 64-bit Mode

In 64-bit mode, the processor does not perform runtime checking on NULL segment selectors. The processor does not cause a #GP fault when an attempt is made to access memory where the referenced segment register has a NULL segment selector.

6.5 PRIVILEGE LEVELS

The processor’s segment-protection mechanism recognizes 4 privilege levels, numbered from 0 to 3. The greater numbers mean lesser privileges. Figure 6-3 shows how these levels of privilege can be interpreted as rings of protection.

The center (reserved for the most privileged code, data, and stacks) is used for the segments containing the critical software, usually the kernel of an operating system. Outer rings are used for less critical software. (Systems that use only 2 of the 4 possible privilege levels should use levels 0 and 3.)

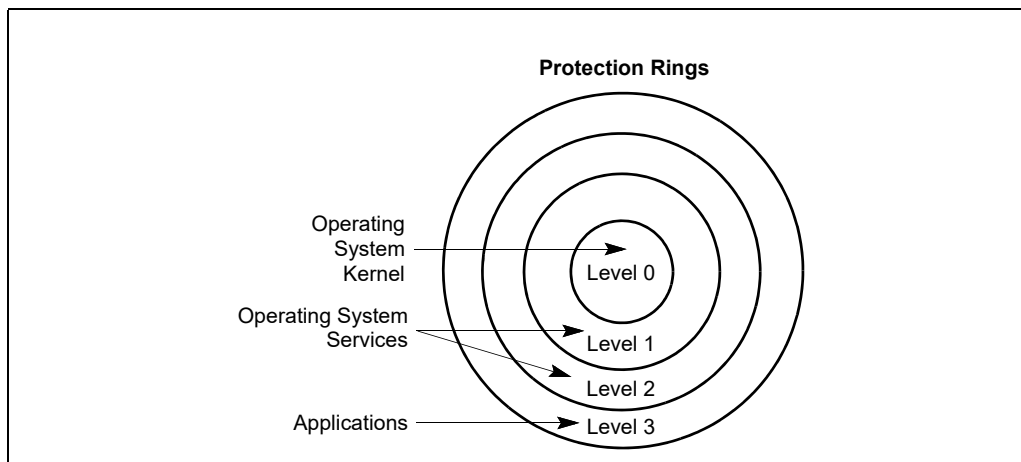


Figure 6-3. Protection Rings

The processor uses privilege levels to prevent a program or task operating at a lesser privilege level from accessing a segment with a greater privilege, except under controlled situations. When the processor detects a privilege level violation, it generates a general-protection exception (#GP).

To carry out privilege-level checks between code segments and data segments, the processor recognizes the following three types of privilege levels:

- **Current privilege level (CPL)** — The CPL is the privilege level of the currently executing program or task. It is stored in bits 0 and 1 of the CS and SS segment registers. Normally, the CPL is equal to the privilege level of the code segment from which instructions are being fetched. The processor changes the CPL when program control is transferred to a code segment with a different privilege level. The CPL is treated slightly differently when accessing conforming code segments. Conforming code segments can be accessed from any privilege level that is equal to or numerically greater (less privileged) than the DPL of the conforming code segment. Also, the CPL is not changed when the processor accesses a conforming code segment that has a different privilege level than the CPL.
- **Descriptor privilege level (DPL)** — The DPL is the privilege level of a segment or gate. It is stored in the DPL field of the segment or gate descriptor for the segment or gate. When the currently executing code segment attempts to access a segment or gate, the DPL of the segment or gate is compared to the CPL and RPL of the segment or gate selector (as described later in this section). The DPL is interpreted differently, depending on the type of segment or gate being accessed:
 - **Data segment** — The DPL indicates the numerically highest privilege level that a program or task can have to be allowed to access the segment. For example, if the DPL of a data segment is 1, only programs running at a CPL of 0 or 1 can access the segment.
 - **Nonconforming code segment (without using a call gate)** — The DPL indicates the privilege level that a program or task must be at to access the segment. For example, if the DPL of a nonconforming code segment is 0, only programs running at a CPL of 0 can access the segment.
 - **Call gate** — The DPL indicates the numerically highest privilege level that the currently executing program or task can be at and still be able to access the call gate. (This is the same access rule as for a data segment.)
 - **Conforming code segment and nonconforming code segment accessed through a call gate** — The DPL indicates the numerically lowest privilege level that a program or task can have to be allowed to access the segment. For example, if the DPL of a conforming code segment is 2, programs running at a CPL of 0 or 1 cannot access the segment.

- **TSS** — The DPL indicates the numerically highest privilege level that the currently executing program or task can be at and still be able to access the TSS. (This is the same access rule as for a data segment.)
- **Requested privilege level (RPL)** — The RPL is an override privilege level that is assigned to segment selectors. It is stored in bits 0 and 1 of the segment selector. The processor checks the RPL along with the CPL to determine if access to a segment is allowed. Even if the program or task requesting access to a segment has sufficient privilege to access the segment, access is denied if the RPL is not of sufficient privilege level. That is, if the RPL of a segment selector is numerically greater than the CPL, the RPL overrides the CPL, and vice versa. The RPL can be used to ensure that privileged code does not access a segment on behalf of an application program unless the program itself has access privileges for that segment. See Section 6.10.4, “Checking Caller Access Privileges (ARPL Instruction),” for a detailed description of the purpose and typical use of the RPL.

Privilege levels are checked when the segment selector of a segment descriptor is loaded into a segment register. The checks used for data access differ from those used for transfers of program control among code segments; therefore, the two kinds of accesses are considered separately in the following sections.

6.6 PRIVILEGE LEVEL CHECKING WHEN ACCESSING DATA SEGMENTS

To access operands in a data segment, the segment selector for the data segment must be loaded into the data-segment registers (DS, ES, FS, or GS) or into the stack-segment register (SS). (Segment registers can be loaded with the MOV, POP, LDS, LES, LFS, LGS, and LSS instructions.) Before the processor loads a segment selector into a segment register, it performs a privilege check (see Figure 6-4) by comparing the privilege levels of the currently running program or task (the CPL), the RPL of the segment selector, and the DPL of the segment’s segment descriptor. The processor loads the segment selector into the segment register if the DPL is numerically greater than or equal to both the CPL and the RPL. Otherwise, a general-protection fault is generated and the segment register is not loaded.

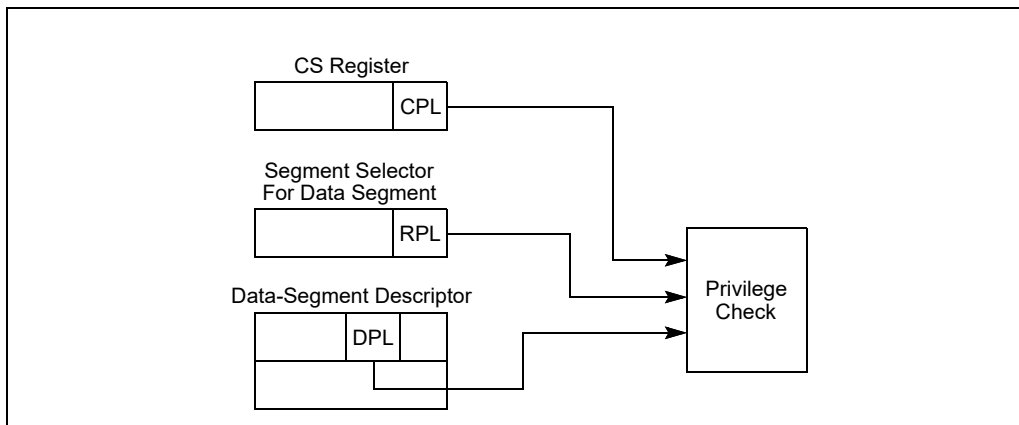


Figure 6-4. Privilege Check for Data Access

Figure 6-5 shows four procedures (located in codes segments A, B, C, and D), each running at different privilege levels and each attempting to access the same data segment.

1. The procedure in code segment A is able to access data segment E using segment selector E1, because the CPL of code segment A and the RPL of segment selector E1 are equal to the DPL of data segment E.
2. The procedure in code segment B is able to access data segment E using segment selector E2, because the CPL of code segment B and the RPL of segment selector E2 are both numerically lower than (more privileged) than the DPL of data segment E. A code segment B procedure can also access data segment E using segment selector E1.
3. The procedure in code segment C is not able to access data segment E using segment selector E3 (dotted line), because the CPL of code segment C and the RPL of segment selector E3 are both numerically greater than (less privileged) than the DPL of data segment E. Even if a code segment C procedure were to use segment selector

E1 or E2, such that the RPL would be acceptable, it still could not access data segment E because its CPL is not privileged enough.

4. The procedure in code segment D should be able to access data segment E because code segment D's CPL is numerically less than the DPL of data segment E. However, the RPL of segment selector E3 (which the code segment D procedure is using to access data segment E) is numerically greater than the DPL of data segment E, so access is not allowed. If the code segment D procedure were to use segment selector E1 or E2 to access the data segment, access would be allowed.

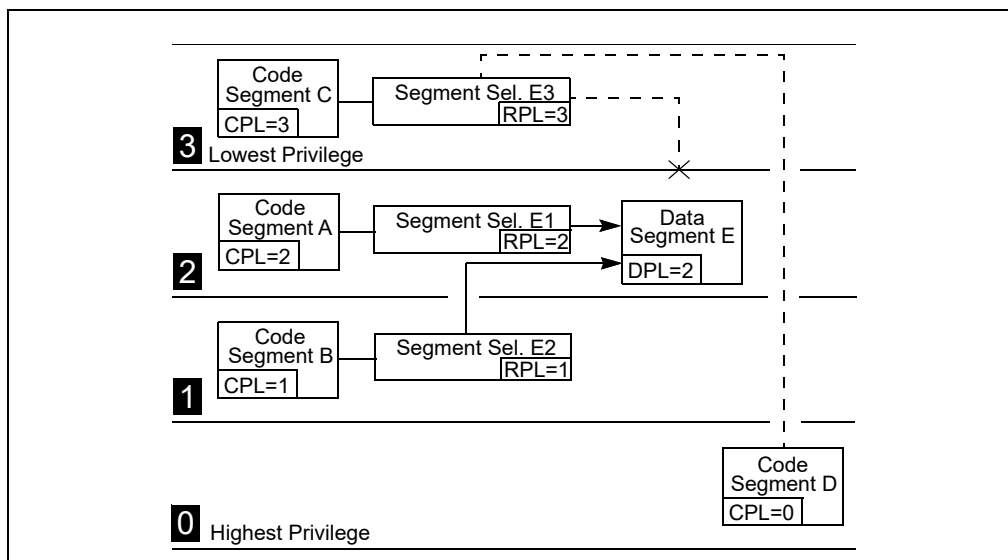


Figure 6-5. Examples of Accessing Data Segments From Various Privilege Levels

As demonstrated in the previous examples, the addressable domain of a program or task varies as its CPL changes. When the CPL is 0, data segments at all privilege levels are accessible; when the CPL is 1, only data segments at privilege levels 1 through 3 are accessible; when the CPL is 3, only data segments at privilege level 3 are accessible.

The RPL of a segment selector can always override the addressable domain of a program or task. When properly used, RPLs can prevent problems caused by accidental (or intentional) use of segment selectors for privileged data segments by less privileged programs or procedures.

It is important to note that the RPL of a segment selector for a data segment is under software control. For example, an application program running at a CPL of 3 can set the RPL for a data-segment selector to 0. With the RPL set to 0, only the CPL checks, not the RPL checks, will provide protection against deliberate, direct attempts to violate privilege-level security for the data segment. To prevent these types of privilege-level-check violations, a program or procedure can check access privileges whenever it receives a data-segment selector from another procedure (see Section 6.10.4, "Checking Caller Access Privileges (ARPL Instruction)").

6.6.1 Accessing Data in Code Segments

In some instances it may be desirable to access data structures that are contained in a code segment. The following methods of accessing data in code segments are possible:

- Load a data-segment register with a segment selector for a nonconforming, readable, code segment.
- Load a data-segment register with a segment selector for a conforming, readable, code segment.
- Use a code-segment override prefix (CS) to read a readable, code segment whose selector is already loaded in the CS register.

The same rules for accessing data segments apply to method 1. Method 2 is always valid because the privilege level of a conforming code segment is effectively the same as the CPL, regardless of its DPL. Method 3 is always valid because the DPL of the code segment selected by the CS register is the same as the CPL.

6.7 PRIVILEGE LEVEL CHECKING WHEN LOADING THE SS REGISTER

Privilege level checking also occurs when the SS register is loaded with the segment selector for a stack segment. Here all privilege levels related to the stack segment must match the CPL; that is, the CPL, the RPL of the stack-segment selector, and the DPL of the stack-segment descriptor must be the same. If the RPL and DPL are not equal to the CPL, a general-protection exception (#GP) is generated.

6.8 PRIVILEGE LEVEL CHECKING WHEN TRANSFERRING PROGRAM CONTROL BETWEEN CODE SEGMENTS

To transfer program control from one code segment to another, the segment selector for the destination code segment must be loaded into the code-segment register (CS). As part of this loading process, the processor examines the segment descriptor for the destination code segment and performs various limit, type, and privilege checks. If these checks are successful, the CS register is loaded, program control is transferred to the new code segment, and program execution begins at the instruction pointed to by the EIP register.

Program control transfers are carried out with the JMP, CALL, RET, SYSENTER, SYSEXIT, SYSCALL, SYSRET, INT *n*, and IRET instructions, as well as by the exception and interrupt mechanisms. Exceptions, interrupts, and the IRET instruction are special cases discussed in Chapter 7, "Interrupt and Exception Handling." This chapter discusses only the JMP, CALL, RET, SYSENTER, SYSEXIT, SYSCALL, and SYSRET instructions.

A JMP or CALL instruction can reference another code segment in any of four ways:

- The target operand contains the segment selector for the target code segment.
- The target operand points to a call-gate descriptor, which contains the segment selector for the target code segment.
- The target operand points to a TSS, which contains the segment selector for the target code segment.
- The target operand points to a task gate, which points to a TSS, which in turn contains the segment selector for the target code segment.

The following sections describe first two types of references. See Section 10.3, "Task Switching," for information on transferring program control through a task gate and/or TSS.

The SYSENTER and SYSEXIT instructions are special instructions for making fast calls to and returns from operating system or executive procedures. These instructions are discussed in Section 6.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."

The SYCALL and SYSRET instructions are special instructions for making fast calls to and returns from operating system or executive procedures in 64-bit mode. These instructions are discussed in Section 6.8.8, "Fast System Calls in 64-Bit Mode."

6.8.1 Direct Calls or Jumps to Code Segments

The near forms of the JMP, CALL, and RET instructions transfer program control within the current code segment, so privilege-level checks are not performed. The far forms of the JMP, CALL, and RET instructions transfer control to other code segments, so the processor does perform privilege-level checks.

When transferring program control to another code segment without going through a call gate, the processor examines four kinds of privilege level and type information (see Figure 6-6):

- The CPL. (Here, the CPL is the privilege level of the calling code segment; that is, the code segment that contains the procedure that is making the call or jump.)

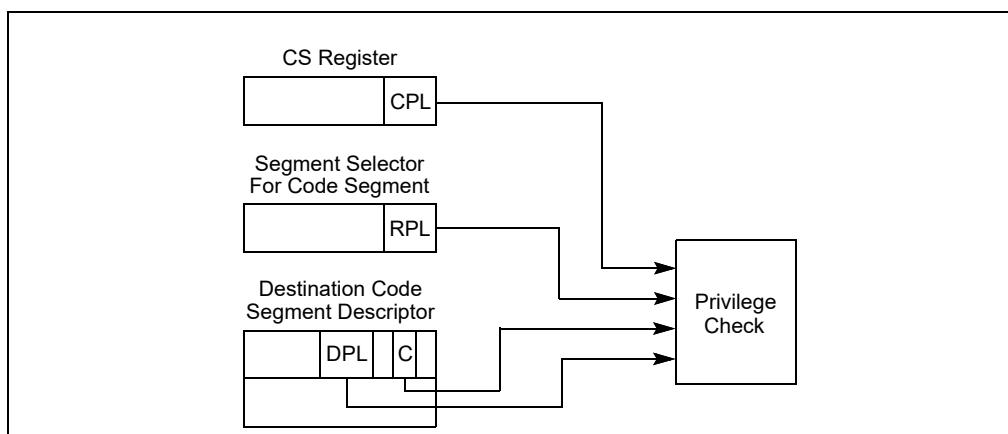


Figure 6-6. Privilege Check for Control Transfer Without Using a Gate

- The DPL of the segment descriptor for the destination code segment that contains the called procedure.
- The RPL of the segment selector of the destination code segment.
- The conforming (C) flag in the segment descriptor for the destination code segment, which determines whether the segment is a conforming (C flag is set) or nonconforming (C flag is clear) code segment. See Section 3.4.5.1, "Code- and Data-Segment Descriptor Types," for more information about this flag.

The rules that the processor uses to check the CPL, RPL, and DPL depends on the setting of the C flag, as described in the following sections.

6.8.1.1 Accessing Nonconforming Code Segments

When accessing nonconforming code segments, the CPL of the calling procedure must be equal to the DPL of the destination code segment; otherwise, the processor generates a general-protection exception (#GP). For example in Figure 6-7:

- Code segment C is a nonconforming code segment. A procedure in code segment A can call a procedure in code segment C (using segment selector C1) because they are at the same privilege level (CPL of code segment A is equal to the DPL of code segment C).
- A procedure in code segment B cannot call a procedure in code segment C (using segment selector C2 or C1) because the two code segments are at different privilege levels.

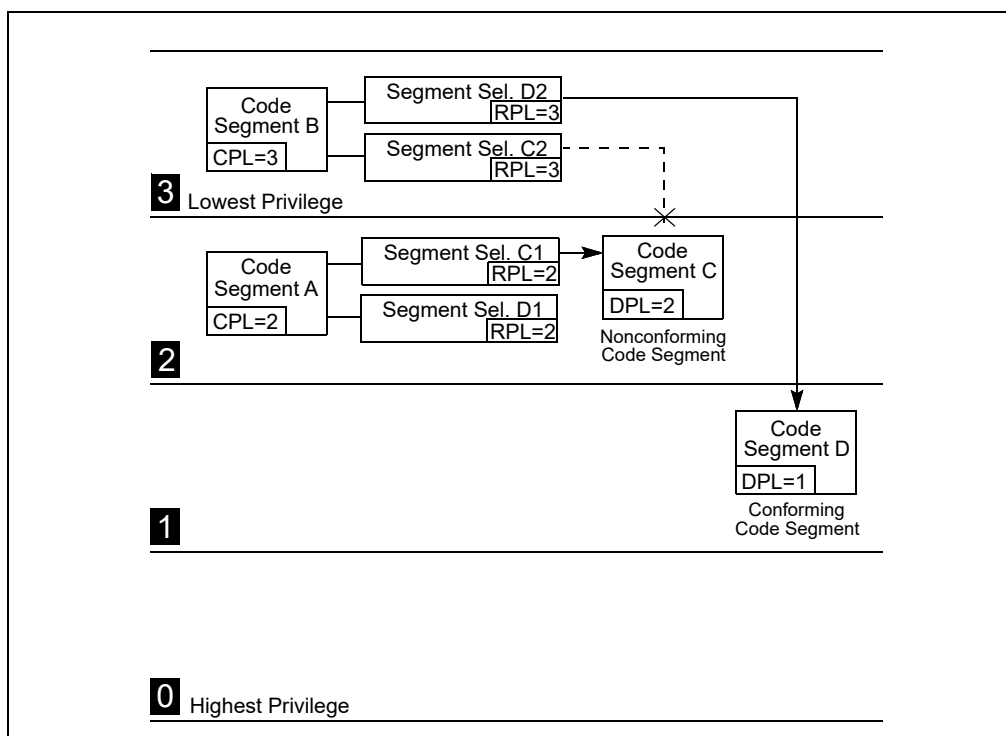


Figure 6-7. Examples of Accessing Conforming and Nonconforming Code Segments From Various Privilege Levels

The RPL of the segment selector that points to a nonconforming code segment has a limited effect on the privilege check. The RPL must be numerically less than or equal to the CPL of the calling procedure for a successful control transfer to occur. So, in the example in Figure 6-7, the RPLs of segment selectors C1 and C2 could legally be set to 0, 1, or 2, but not to 3.

When the segment selector of a nonconforming code segment is loaded into the CS register, the privilege level field is not changed; that is, it remains at the CPL (which is the privilege level of the calling procedure). This is true, even if the RPL of the segment selector is different from the CPL.

6.8.1.2 Accessing Conforming Code Segments

When accessing conforming code segments, the CPL of the calling procedure may be numerically equal to or greater than (less privileged) the DPL of the destination code segment; the processor generates a general-protection exception (#GP) only if the CPL is less than the DPL. (The segment selector RPL for the destination code segment is not checked if the segment is a conforming code segment.)

In the example in Figure 6-7, code segment D is a conforming code segment. Therefore, calling procedures in both code segment A and B can access code segment D (using either segment selector D1 or D2, respectively), because they both have CPLs that are greater than or equal to the DPL of the conforming code segment. **For conforming code segments, the DPL represents the numerically lowest privilege level that a calling procedure may be at to successfully make a call to the code segment.**

(Note that segments selectors D1 and D2 are identical except for their respective RPLs. But since RPLs are not checked when accessing conforming code segments, the two segment selectors are essentially interchangeable.)

When program control is transferred to a conforming code segment, the CPL does not change, even if the DPL of the destination code segment is less than the CPL. This situation is the only one where the CPL may be different from the DPL of the current code segment. Also, since the CPL does not change, no stack switch occurs.

Conforming segments are used for code modules such as math libraries and exception handlers, which support applications but do not require access to protected system facilities. These modules are part of the operating system or executive software, but they can be executed at numerically higher privilege levels (less privileged levels). Keeping the CPL at the level of a calling code segment when switching to a conforming code segment

prevents an application program from accessing nonconforming code segments while at the privilege level (DPL) of a conforming code segment and thus prevents it from accessing more privileged data.

Most code segments are nonconforming. For these segments, program control can be transferred only to code segments at the same level of privilege, unless the transfer is carried out through a call gate, as described in the following sections.

6.8.2 Gate Descriptors

To provide controlled access to code segments with different privilege levels, the processor provides special set of descriptors called gate descriptors. There are four kinds of gate descriptors:

- Call gates
- Trap gates
- Interrupt gates
- Task gates

Task gates are used for task switching and are discussed in Chapter 10, “Task Management.” Trap and interrupt gates are special kinds of call gates used for calling exception and interrupt handlers. The are described in Chapter 7, “Interrupt and Exception Handling.” This chapter is concerned only with call gates.

6.8.3 Call Gates

Call gates facilitate controlled transfers of program control between different privilege levels. They are typically used only in operating systems or executives that use the privilege-level protection mechanism. Call gates are also useful for transferring program control between 16-bit and 32-bit code segments, as described in Section 24.4, “Transferring Control Among Mixed-Size Code Segments.”

Figure 6-8 shows the format of a call-gate descriptor. A call-gate descriptor may reside in the GDT or in an LDT, but not in the interrupt descriptor table (IDT). It performs six functions:

- It specifies the code segment to be accessed.
- It defines an entry point for a procedure in the specified code segment.
- It specifies the privilege level required for a caller trying to access the procedure.

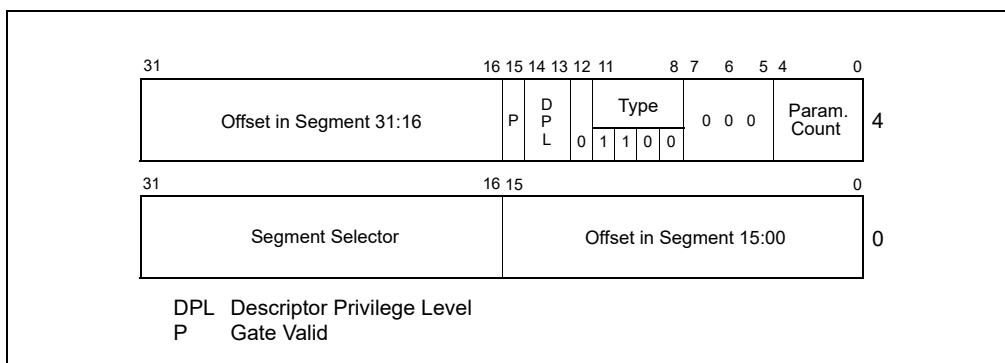


Figure 6-8. Call-Gate Descriptor

- If a stack switch occurs, it specifies the number of optional parameters to be copied between stacks.
- It defines the size of values to be pushed onto the target stack: 16-bit gates force 16-bit pushes and 32-bit gates force 32-bit pushes.
- It specifies whether the call-gate descriptor is valid.

The segment selector field in a call gate specifies the code segment to be accessed. The offset field specifies the entry point in the code segment. This entry point is generally to the first instruction of a specific procedure. The DPL field indicates the privilege level of the call gate, which in turn is the privilege level required to access the selected

procedure through the gate. The P flag indicates whether the call-gate descriptor is valid. (The presence of the code segment to which the gate points is indicated by the P flag in the code segment’s descriptor.) The parameter count field indicates the number of parameters to copy from the calling procedures stack to the new stack if a stack switch occurs (see Section 6.8.5, “Stack Switching”). The parameter count specifies the number of words for 16-bit call gates and doublewords for 32-bit call gates.

Note that the P flag in a gate descriptor is normally always set to 1. If it is set to 0, a not present (#NP) exception is generated when a program attempts to access the descriptor. The operating system can use the P flag for special purposes. For example, it could be used to track the number of times the gate is used. Here, the P flag is initially set to 0 causing a trap to the not-present exception handler. The exception handler then increments a counter and sets the P flag to 1, so that on returning from the handler, the gate descriptor will be valid.

6.8.3.1 IA-32e Mode Call Gates

Call-gate descriptors in 32-bit mode provide a 32-bit offset for the instruction pointer (EIP); 64-bit extensions double the size of 32-bit mode call gates in order to store 64-bit instruction pointers (RIP). See Figure 6-9:

- The first eight bytes (bytes 7:0) of a 64-bit mode call gate are similar but not identical to legacy 32-bit mode call gates. The parameter-copy-count field has been removed.
- Bytes 11:8 hold the upper 32 bits of the target-segment offset in canonical form. A general-protection exception (#GP) is generated if software attempts to use a call gate with a target offset that is not in canonical form.
- 16-byte descriptors may reside in the same descriptor table with 16-bit and 32-bit descriptors. A type field, used for consistency checking, is defined in bits 12:8 of the 64-bit descriptor’s highest dword (cleared to zero). A general-protection exception (#GP) results if an attempt is made to access the upper half of a 64-bit mode descriptor as a 32-bit mode descriptor.

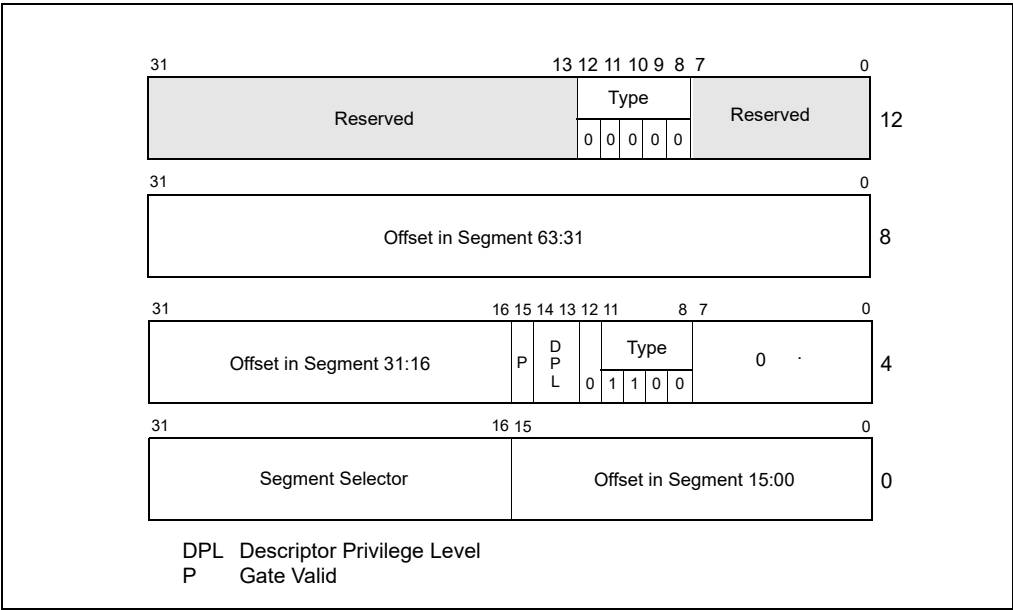


Figure 6-9. Call-Gate Descriptor in IA-32e Mode

- Target code segments referenced by a 64-bit call gate must be 64-bit code segments (CS.L = 1, CS.D = 0). If not, the reference generates a general-protection exception, #GP (CS selector).
- Only 64-bit mode call gates can be referenced in IA-32e mode (64-bit mode and compatibility mode). The legacy 32-bit mode call gate type (0CH) is redefined in IA-32e mode as a 64-bit call-gate type; no 32-bit call-gate type exists in IA-32e mode.

- If a far call references a 16-bit call gate type (04H) in IA-32e mode, a general-protection exception (#GP) is generated.

When a call references a 64-bit mode call gate, actions taken are identical to those taken in 32-bit mode, with the following exceptions:

- Stack pushes are made in eight-byte increments.
- A 64-bit RIP is pushed onto the stack.
- Parameter copying is not performed.

Use a matching far-return instruction size for correct operation (returns from 64-bit calls must be performed with a 64-bit operand-size return to process the stack correctly).

Call gates cannot be used when FRED transitions are enabled. An execution of CALL or JMP that accesses a call gate when FRED transitions are enabled will cause a general-protection exception (#GP).

6.8.4 Accessing a Code Segment Through a Call Gate

To access a call gate, a far pointer to the gate is provided as a target operand in a CALL or JMP instruction. The segment selector from this pointer identifies the call gate (see Figure 6-10); the offset from the pointer is required, but not used or checked by the processor. (The offset can be set to any value.)

When the processor has accessed the call gate, it uses the segment selector from the call gate to locate the segment descriptor for the destination code segment. (This segment descriptor can be in the GDT or the LDT.) It then combines the base address from the code-segment descriptor with the offset from the call gate to form the linear address of the procedure entry point in the code segment.

As shown in Figure 6-11, four different privilege levels are used to check the validity of a program control transfer through a call gate:

- The CPL (current privilege level).
- The RPL (requestor privilege level) of the call gate's selector.
- The DPL (descriptor privilege level) of the call gate descriptor.
- The DPL of the segment descriptor of the destination code segment.

The C flag (conforming) in the segment descriptor for the destination code segment is also checked.

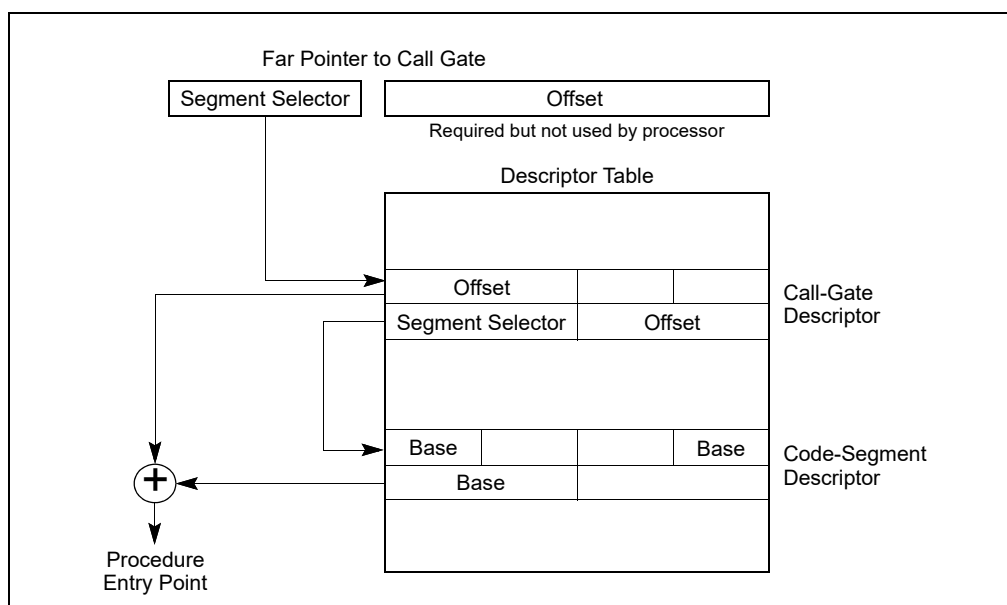


Figure 6-10. Call-Gate Mechanism

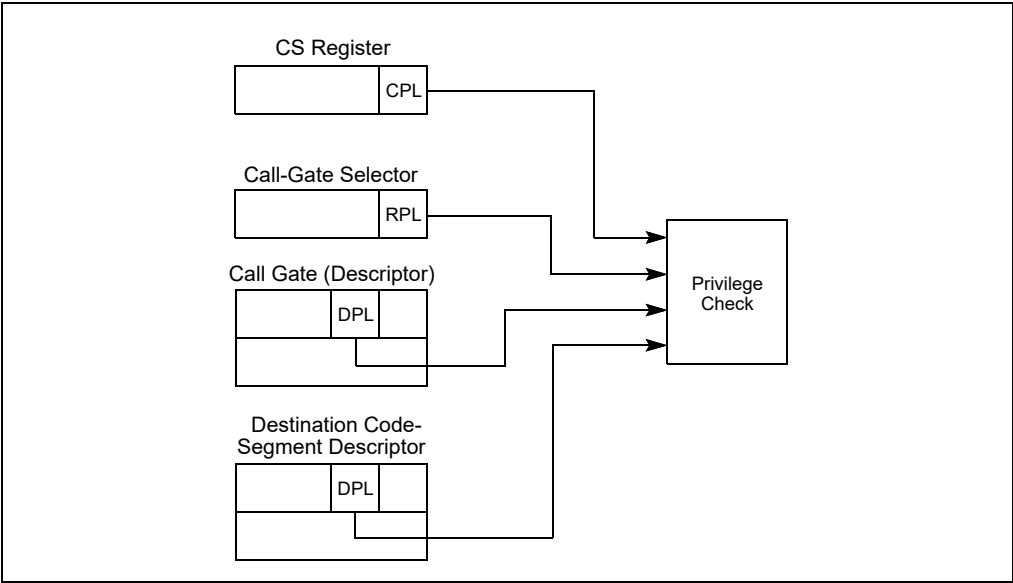


Figure 6-11. Privilege Check for Control Transfer with Call Gate

The privilege checking rules are different depending on whether the control transfer was initiated with a CALL or a JMP instruction, as shown in Table 6-1.

Table 6-1. Privilege Check Rules for Call Gates

Instruction	Privilege Check Rules
CALL	$CPL \leq \text{call gate DPL}$; $RPL \leq \text{call gate DPL}$ Destination conforming code segment $DPL \leq CPL$ Destination nonconforming code segment $DPL \leq CPL$
JMP	$CPL \leq \text{call gate DPL}$; $RPL \leq \text{call gate DPL}$ Destination conforming code segment $DPL \leq CPL$ Destination nonconforming code segment $DPL = CPL$

The DPL field of the call-gate descriptor specifies the numerically highest privilege level from which a calling procedure can access the call gate; that is, to access a call gate, the CPL of a calling procedure must be equal to or less than the DPL of the call gate. For example, in Figure 6-15, call gate A has a DPL of 3. So calling procedures at all CPLs (0 through 3) can access this call gate, which includes calling procedures in code segments A, B, and C. Call gate B has a DPL of 2, so only calling procedures at a CPL of 0, 1, or 2 can access call gate B, which includes calling procedures in code segments B and C. The dotted line shows that a calling procedure in code segment A cannot access call gate B.

The RPL of the segment selector to a call gate must satisfy the same test as the CPL of the calling procedure; that is, the RPL must be less than or equal to the DPL of the call gate. In the example in Figure 6-15, a calling procedure in code segment C can access call gate B using gate selector B2 or B1, but it could not use gate selector B3 to access call gate B.

If the privilege checks between the calling procedure and call gate are successful, the processor then checks the DPL of the code-segment descriptor against the CPL of the calling procedure. Here, the privilege check rules vary between CALL and JMP instructions. Only CALL instructions can use call gates to transfer program control to more privileged (numerically lower privilege level) nonconforming code segments; that is, to nonconforming code segments with a DPL less than the CPL. A JMP instruction can use a call gate only to transfer program control to a nonconforming code segment with a DPL equal to the CPL. CALL and JMP instruction can both transfer program control to a more privileged conforming code segment; that is, to a conforming code segment with a DPL less than or equal to the CPL.

If a call is made to a more privileged (numerically lower privilege level) nonconforming destination code segment, the CPL is lowered to the DPL of the destination code segment and a stack switch occurs (see Section 6.8.5, “Stack Switching”). If a call or jump is made to a more privileged conforming destination code segment, the CPL is not changed and no stack switch occurs.

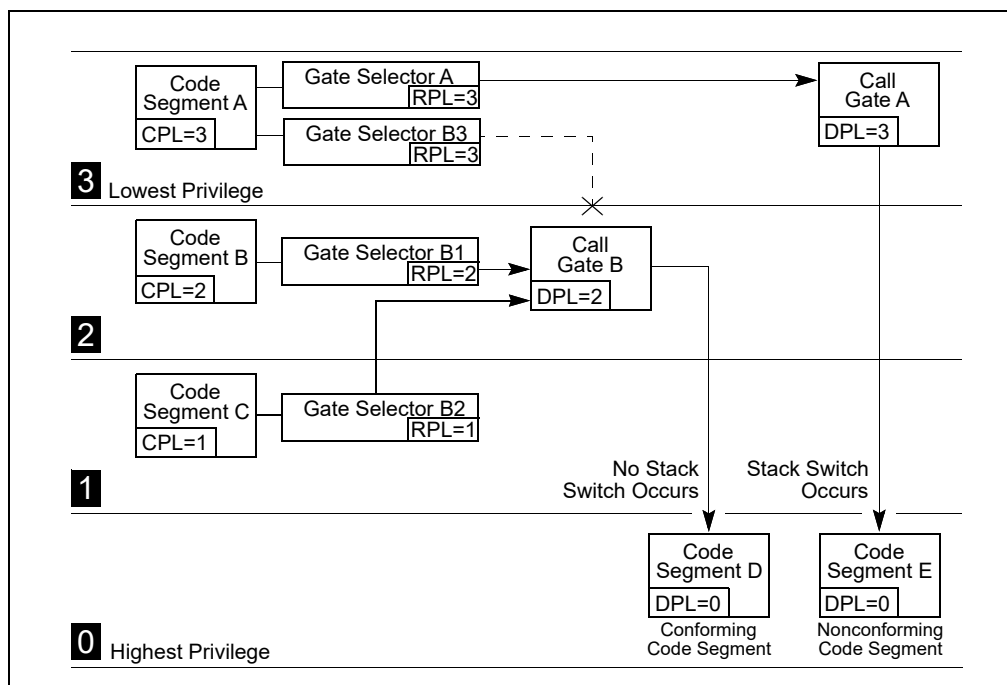


Figure 6-12. Example of Accessing Call Gates At Various Privilege Levels

Call gates allow a single code segment to have procedures that can be accessed at different privilege levels. For example, an operating system located in a code segment may have some services which are intended to be used by both the operating system and application software (such as procedures for handling character I/O). Call gates for these procedures can be set up that allow access at all privilege levels (0 through 3). More privileged call gates (with DPLs of 0 or 1) can then be set up for other operating system services that are intended to be used only by the operating system (such as procedures that initialize device drivers).

6.8.5 Stack Switching

Whenever a call gate is used to transfer program control to a more privileged nonconforming code segment (that is, when the DPL of the nonconforming destination code segment is less than the CPL), the processor automatically switches to the stack for the destination code segment’s privilege level. This stack switching is carried out to prevent more privileged procedures from crashing due to insufficient stack space. It also prevents less privileged procedures from interfering (by accident or intent) with more privileged procedures through a shared stack.

Each task must define up to 4 stacks: one for applications code (running at privilege level 3) and one for each of the privilege levels 2, 1, and 0 that are used. (If only two privilege levels are used [3 and 0], then only two stacks must be defined.) Each of these stacks is located in a separate segment and is identified with a segment selector and an offset into the stack segment (a stack pointer).

The segment selector and stack pointer for the privilege level 3 stack is located in the SS and ESP registers, respectively, when privilege-level-3 code is being executed and is automatically stored on the called procedure’s stack when a stack switch occurs.

Pointers to the privilege level 0, 1, and 2 stacks are stored in the TSS for the currently running task (see Figure 10-2). Each of these pointers consists of a segment selector and a stack pointer (loaded into the ESP register). These initial pointers are strictly read-only values. The processor does not change them while the task is running. They are used only to create new stacks when calls are made to more privileged levels (numerically lower

privilege levels). These stacks are disposed of when a return is made from the called procedure. The next time the procedure is called, a new stack is created using the initial stack pointer. (The TSS does not specify a stack for privilege level 3 because the processor does not allow a transfer of program control from a procedure running at a CPL of 0, 1, or 2 to a procedure running at a CPL of 3, except on a return.)

The operating system is responsible for creating stacks and stack-segment descriptors for all the privilege levels to be used and for loading initial pointers for these stacks into the TSS. Each stack must be read/write accessible (as specified in the type field of its segment descriptor) and must contain enough space (as specified in the limit field) to hold the following items:

- The contents of the SS, ESP, CS, and EIP registers for the calling procedure.
- The parameters and temporary variables required by the called procedure.
- The EFLAGS register and error code, when implicit calls are made to an exception or interrupt handler.

The stack will need to require enough space to contain many frames of these items, because procedures often call other procedures, and an operating system may support nesting of multiple interrupts. Each stack should be large enough to allow for the worst case nesting scenario at its privilege level.

(If the operating system does not use the processor's multitasking mechanism, it still must create at least one TSS for this stack-related purpose.)

When a procedure call through a call gate results in a change in privilege level, the processor performs the following steps to switch stacks and begin execution of the called procedure at a new privilege level:

1. Uses the DPL of the destination code segment (the new CPL) to select a pointer to the new stack (segment selector and stack pointer) from the TSS.
2. Reads the segment selector and stack pointer for the stack to be switched to from the current TSS. Any limit violations detected while reading the stack-segment selector, stack pointer, or stack-segment descriptor cause an invalid TSS (#TS) exception to be generated.
3. Checks the stack-segment descriptor for the proper privileges and type and generates an invalid TSS (#TS) exception if violations are detected.
4. Temporarily saves the current values of the SS and ESP registers.
5. Loads the segment selector and stack pointer for the new stack in the SS and ESP registers.
6. Pushes the temporarily saved values for the SS and ESP registers (for the calling procedure) onto the new stack (see Figure 6-13).
7. Copies the number of parameter specified in the parameter count field of the call gate from the calling procedure's stack to the new stack. If the count is 0, no parameters are copied.
8. Pushes the return instruction pointer (the current contents of the CS and EIP registers) onto the new stack.
9. Loads the segment selector for the new code segment and the new instruction pointer from the call gate into the CS and EIP registers, respectively, and begins execution of the called procedure.

See the description of the CALL instruction in Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for a detailed description of the privilege level checks and other protection checks that the processor performs on a far call through a call gate.

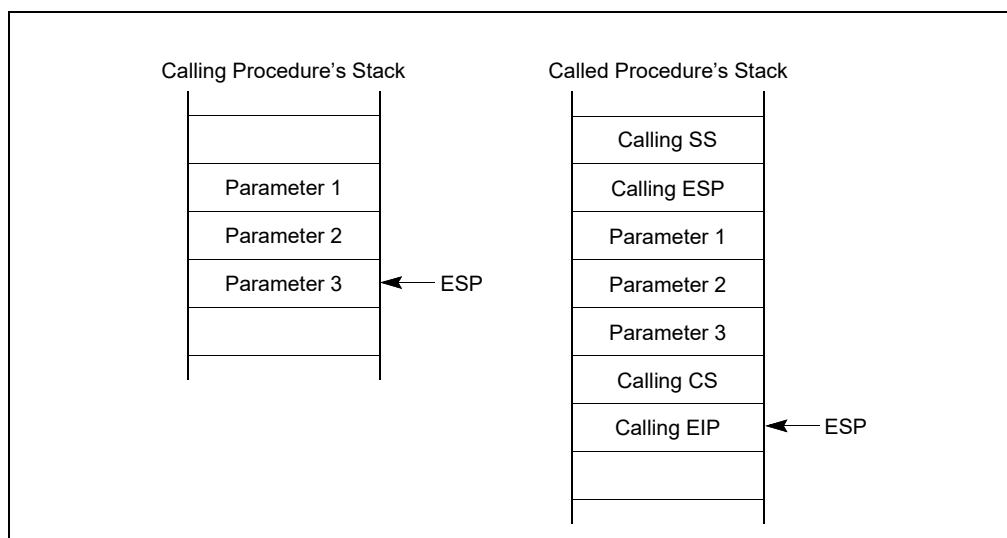


Figure 6-13. Stack Switching During an Interprivilege-Level Call

The parameter count field in a call gate specifies the number of data items (up to 31) that the processor should copy from the calling procedure's stack to the stack of the called procedure. If more than 31 data items need to be passed to the called procedure, one of the parameters can be a pointer to a data structure, or the saved contents of the SS and ESP registers may be used to access parameters in the old stack space. The size of the data items passed to the called procedure depends on the call gate size, as described in Section 6.8.3, "Call Gates."

6.8.5.1 Stack Switching in 64-bit Mode

Although protection-check rules for call gates are unchanged from 32-bit mode, stack-switch changes in 64-bit mode are different.

When stacks are switched as part of a 64-bit mode privilege-level change through a call gate, a new SS (stack segment) descriptor is not loaded; 64-bit mode only loads an inner-level RSP from the TSS. The new SS is forced to NULL and the SS selector's RPL field is forced to the new CPL. The new SS is set to NULL in order to handle nested far transfers (far CALL, INTn, interrupts, and exceptions). The old SS and RSP are saved on the new stack.

On a subsequent far RET, the old SS is popped from the stack and loaded into the SS register. See Table 6-2.

Table 6-2. 64-Bit-Mode Stack Layout After Far CALL with CPL Change

32-bit Mode		ESP	IA-32e mode		RSP
Old SS Selector	+12		+24	Old SS Selector	
Old ESP	+8		+16	Old RSP	
CS Selector	+4		+8	Old CS Selector	
EIP	0		0	RIP	
< 4 Bytes >				< 8 Bytes >	

In 64-bit mode, stack operations resulting from a privilege-level-changing far call or far return are eight-bytes wide and change the RSP by eight. The mode does not support the automatic parameter-copy feature found in 32-bit mode. The call-gate count field is ignored. Software can access the old stack, if necessary, by referencing the old stack-segment selector and stack pointer saved on the new process stack.

In 64-bit mode, far RET is allowed to load a NULL SS under certain conditions. If the target mode is 64-bit mode and the target CPL $\neq$ 3, IRET allows SS to be loaded with a NULL selector. If the called procedure itself is interrupted, the NULL SS is pushed on the stack frame. On the subsequent far RET, the NULL SS on the stack acts as a flag to tell the processor not to load a new SS descriptor.

6.8.6 Returning from a Called Procedure

The RET instruction can be used to perform a near return, a far return at the same privilege level, and a far return to a different privilege level. This instruction is intended to execute returns from procedures that were called with a CALL instruction. It does not support returns from a JMP instruction, because the JMP instruction does not save a return instruction pointer on the stack.

A near return only transfers program control within the current code segment; therefore, the processor performs only a limit check. When the processor pops the return instruction pointer from the stack into the EIP register, it checks that the pointer does not exceed the limit of the current code segment.

On a far return at the same privilege level, the processor pops both a segment selector for the code segment being returned to and a return instruction pointer from the stack. Under normal conditions, these pointers should be valid, because they were pushed on the stack by the CALL instruction. However, the processor performs privilege checks to detect situations where the current procedure might have altered the pointer or failed to maintain the stack properly.

A far return that requires a privilege-level change is only allowed when returning to a less privileged level (that is, the DPL of the return code segment is numerically greater than the CPL). The processor uses the RPL field from the CS register value saved for the calling procedure (see Figure 6-13) to determine if a return to a numerically higher privilege level is required. If the RPL is numerically greater (less privileged) than the CPL, a return across privilege levels occurs.

The processor performs the following steps when performing a far return to a calling procedure (see Figures 6-2 and 6-4 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for an illustration of the stack contents prior to and after a return):

1. Checks the RPL field of the saved CS register value to determine if a privilege level change is required on the return.
2. Loads the CS and EIP registers with the values on the called procedure's stack. (Type and privilege level checks are performed on the code-segment descriptor and RPL of the code-segment selector.)
3. (If the RET instruction includes a parameter count operand and the return requires a privilege level change.) Adds the parameter count (in bytes obtained from the RET instruction) to the current ESP register value (after popping the CS and EIP values), to step past the parameters on the called procedure's stack. The resulting value in the ESP register points to the saved SS and ESP values for the calling procedure's stack. (Note that the byte count in the RET instruction must be chosen to match the parameter count in the call gate that the calling procedure referenced when it made the original call multiplied by the size of the parameters.)
4. (If the return requires a privilege level change.) Loads the SS and ESP registers with the saved SS and ESP values and switches back to the calling procedure's stack. The SS and ESP values for the called procedure's stack are discarded. Any limit violations detected while loading the stack-segment selector or stack pointer cause a general-protection exception (#GP) to be generated. The new stack-segment descriptor is also checked for type and privilege violations.
5. (If the RET instruction includes a parameter count operand.) Adds the parameter count (in bytes obtained from the RET instruction) to the current ESP register value, to step past the parameters on the calling procedure's stack. The resulting ESP value is not checked against the limit of the stack segment. If the ESP value is beyond the limit, that fact is not recognized until the next stack operation.
6. (If the return requires a privilege level change.) Checks the contents of the DS, ES, FS, and GS segment registers. If any of these registers refer to segments whose DPL is less than the new CPL (excluding conforming code segments), the segment register is loaded with a null segment selector.

See the description of the RET instruction in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B, for a detailed description of the privilege level checks and other protection checks that the processor performs on a far return.

6.8.7 Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions

The SYSENTER and SYSEXIT instructions were introduced into the IA-32 architecture in the Pentium II processors for the purpose of providing a fast (low overhead) mechanism for calling operating system or executive procedures.

SYSENTER is intended for use by user code running at privilege level 3 to access operating system or executive procedures running at privilege level 0. SYSEXIT is intended for use by privilege level 0 operating system or executive procedures for fast returns to privilege level 3 user code. SYSENTER can be executed from privilege levels 3, 2, 1, or 0; SYSEXIT can only be executed from privilege level 0.

The SYSENTER and SYSEXIT instructions are companion instructions, but they do not constitute a call/return pair. This is because SYSENTER does not save any state information for use by SYSEXIT on a return.

The target instruction and stack pointer for these instructions are not specified through instruction operands. Instead, they are specified through parameters entered in MSRs and general-purpose registers.

For SYSENTER, target fields are generated using the following sources:

- **Target code segment** — Reads this from IA32_SYSENTER_CS.
- **Target instruction** — Reads this from IA32_SYSENTER_EIP.
- **Stack segment** — Computed by adding 8 to the value in IA32_SYSENTER_CS.
- **Stack pointer** — Reads this from the IA32_SYSENTER_ESP.

For SYSEXIT, target fields are generated using the following sources:

- **Target code segment** — Computed by adding 16 to the value in the IA32_SYSENTER_CS.
- **Target instruction** — Reads this from EDI.
- **Stack segment** — Computed by adding 24 to the value in IA32_SYSENTER_CS.
- **Stack pointer** — Reads this from ECX.

The SYSENTER and SYSEXIT instructions perform “fast” calls and returns because they force the processor into a predefined privilege level 0 state when SYSENTER is executed and into a predefined privilege level 3 state when SYSEXIT is executed. By forcing predefined and consistent processor states, the number of privilege checks ordinarily required to perform a far call to another privilege levels are greatly reduced. Also, by predefining the target context state in MSRs and general-purpose registers eliminates all memory accesses except when fetching the target code.

Any additional state that needs to be saved to allow a return to the calling procedure must be saved explicitly by the calling procedure or be predefined through programming conventions.

6.8.7.1 SYSENTER and SYSEXIT Instructions in IA-32e Mode

For Intel 64 processors, the SYSENTER and SYSEXIT instructions are enhanced to allow fast system calls from user code running at privilege level 3 (in compatibility mode or 64-bit mode) to 64-bit executive procedures running at privilege level 0. IA32_SYSENTER_EIP MSR and IA32_SYSENTER_ESP MSR are expanded to hold 64-bit addresses. If IA-32e mode is inactive, only the lower 32-bit addresses stored in these MSRs are used. The WRMSR instruction ensures that the addresses stored in these MSRs are canonical. Note that, in 64-bit mode, IA32_SYSENTER_CS must not contain a NULL selector.

When SYSENTER transfers control, the following fields are generated and bits set:

- **Target code segment** — Reads non-NUL selector from IA32_SYSENTER_CS.
- **New CS attributes** — CS base = 0, CS limit = FFFFFFFFH.
- **Target instruction** — Reads 64-bit canonical address from IA32_SYSENTER_EIP.
- **Stack segment** — Computed by adding 8 to the value from IA32_SYSENTER_CS.
- **Stack pointer** — Reads 64-bit canonical address from IA32_SYSENTER_ESP.
- **New SS attributes** — SS base = 0, SS limit = FFFFFFFFH.

When the SYSEXIT instruction transfers control to 64-bit mode user code using REX.W, the following fields are generated and bits set:

- **Target code segment** — Computed by adding 32 to the value in IA32_SYSENTER_CS.
- **New CS attributes** — L-bit = 1 (go to 64-bit mode).
- **Target instruction** — Reads 64-bit canonical address in RDI.
- **Stack segment** — Computed by adding 40 to the value of IA32_SYSENTER_CS.

- **Stack pointer** — Update RSP using 64-bit canonical address in RCX.

When SYSEXIT transfers control to compatibility mode user code when the operand size attribute is 32 bits, the following fields are generated and bits set:

- **Target code segment** — Computed by adding 16 to the value in IA32_SYSENTER_CS.
- **New CS attributes** — L-bit = 0 (go to compatibility mode).
- **Target instruction** — Fetch the target instruction from 32-bit address in EDX.
- **Stack segment** — Computed by adding 24 to the value in IA32_SYSENTER_CS.
- **Stack pointer** — Update ESP from 32-bit address in ECX.

6.8.8 Fast System Calls in 64-Bit Mode

The SYSCALL and SYSRET instructions are designed for operating systems that use a flat memory model (segmentation is not used). The instructions, along with SYSENTER and SYSEXIT, are suited for IA-32e mode operation. SYSCALL and SYSRET, however, are not supported in compatibility mode (or in protected mode). Use CPUID to check if SYSCALL and SYSRET are available (CPUID.80000001H:EDX[11] = 1).

SYSCALL is intended for use by user code running at privilege level 3 to access operating system or executive procedures running at privilege level 0. SYSRET is intended for use by privilege level 0 operating system or executive procedures for fast returns to privilege level 3 user code.

Stack pointers for SYSCALL/SYSRET are not specified through model specific registers. The clearing of bits in RFLAGS is programmable rather than fixed. SYSCALL/SYSRET save and restore the RFLAGS register.

For SYSCALL, the processor saves RFLAGS into R11 and the RIP of the next instruction into RCX; it then gets the privilege-level 0 target code segment, instruction pointer, stack segment, and flags as follows:

- **Target code segment** — Reads a non-NULL selector from IA32_STAR[47:32].
- **Target instruction pointer** — Reads a 64-bit address from IA32_LSTAR. (The WRMSR instruction ensures that the value of the IA32_LSTAR MSR is canonical.)
- **Stack segment** — Computed by adding 8 to the value in IA32_STAR[47:32].
- **Flags** — The processor sets RFLAGS to the logical-AND of its current value with the complement of the value in the IA32_FMASK MSR.

When SYSRET transfers control to 64-bit mode user code using REX.W, the processor gets the privilege level 3 target code segment, instruction pointer, stack segment, and flags as follows:

- **Target code segment** — Reads a non-NULL selector from IA32_STAR[63:48] + 16.
- **Target instruction pointer** — Copies the value in RCX into RIP.
- **Stack segment** — IA32_STAR[63:48] + 8.
- **EFLAGS** — Loaded from R11.

When SYSRET transfers control to 32-bit mode user code using a 32-bit operand size, the processor gets the privilege level 3 target code segment, instruction pointer, stack segment, and flags as follows:

- **Target code segment** — Reads a non-NULL selector from IA32_STAR[63:48].
- **Target instruction pointer** — Copies the value in ECX into EIP.
- **Stack segment** — IA32_STAR[63:48] + 8.
- **EFLAGS** — Loaded from R11.

It is the responsibility of the OS to ensure the descriptors in the GDT/LDT correspond to the selectors loaded by SYSCALL/SYSRET (consistent with the base, limit, and attribute values forced by the instructions).

See Figure 6-14 for the layout of IA32_STAR, IA32_LSTAR, and IA32_FMASK.

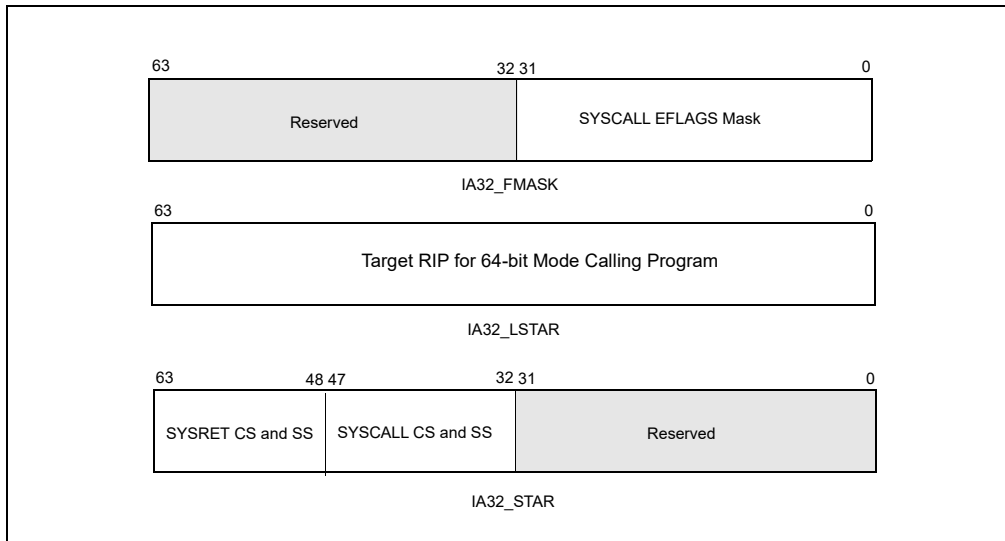


Figure 6-14. MSRs Used by SYSCALL and SYSRET

The SYSCALL instruction does not save the stack pointer, and the SYSRET instruction does not restore it. It is likely that the OS system-call handler will change the stack pointer from the user stack to the OS stack. If so, it is the responsibility of software first to save the user stack pointer. This might be done by user code, prior to executing SYSCALL, or by the OS system-call handler after SYSCALL.

Because the SYSRET instruction does not modify the stack pointer, it is necessary for software to switch back to the user stack. The OS may load the user stack pointer (if it was saved after SYSCALL) before executing SYSRET; alternatively, user code may load the stack pointer (if it was saved before SYSCALL) after receiving control from SYSRET.

If the OS loads the stack pointer before executing SYSRET, it must ensure that the handler of any interrupt or exception delivered between restoring the stack pointer and successful execution of SYSRET is not invoked with the user stack. It can do so using approaches such as the following:

- External interrupts. The OS can prevent an external interrupt from being delivered by clearing EFLAGS.IF before loading the user stack pointer.
- Nonmaskable interrupts (NMIs). The OS can ensure that the NMI handler is invoked with the correct stack by using the interrupt stack table (IST) mechanism for gate 2 (NMI) in the IDT (see Section 7.14.5, "Interrupt Stack Table").
- General-protection exceptions (#GP). The SYSRET instruction generates #GP(0) if the value of RCX is not canonical. The OS can address this possibility using one or more of the following approaches:
 - Confirming that the value of RCX is canonical before executing SYSRET.
 - Using paging to ensure that the SYSCALL instruction will never save a non-canonical value into RCX.
 - Using the IST mechanism for gate 13 (#GP) in the IDT.

When FRED transitions are enabled, SYSCALL and SYSENTER are implemented using FRED event delivery. SYSRET and SYSEXIT are not used; instead, ERETU (or ERETS) are used to return. See Chapter 8, "Flexible Return and Event Delivery (FRED)."

6.9 PRIVILEGED INSTRUCTIONS

Some of the system instructions (called "privileged instructions") are protected from use by application programs. The privileged instructions control system functions (such as the loading of system registers). They can be

executed only when the CPL is 0 (most privileged). If one of these instructions is executed when the CPL is not 0, a general-protection exception (#GP) is generated. The following system instructions are privileged instructions:

- LGDT — Load GDT register.
- LLDT — Load LDT register.
- LTR — Load task register.
- LIDT — Load IDT register.
- MOV (control registers) — Load and store control registers.
- LMSW — Load machine status word.
- CLTS — Clear task-switched flag in register CR0.
- MOV (debug registers) — Load and store debug registers.
- INVD — Invalidate cache, without writeback.
- WBINVD — Invalidate cache, with writeback.
- INVLPG — Invalidate TLB entry.
- HLT — Halt processor.
- RDMSR, [RDMSRLIST](#) — Read Model-Specific Registers.
- WRMSR, [WRMSRLIST](#), [WRMSRNS](#) — Write Model-Specific Registers.
- RDPMC — Read Performance-Monitoring Counter.
- RDTSC — Read Time-Stamp Counter.
- [ERETS](#) — Event return to supervisor
- [ERETU](#) — Event return to user
- [LKGS](#) — Load kernel GS base

Some of the privileged instructions are available only in the more recent families of Intel 64 and IA-32 processors (see Section 25.13, “New Instructions In the Pentium and Later IA-32 Processors”).

The PCE and TSD flags in register CR4 (bits 4 and 2, respectively) enable the RDPMC and RDTSC instructions, respectively, to be executed at any CPL.

6.10 POINTER VALIDATION

When operating in protected mode, the processor validates all pointers to enforce protection between segments and maintain isolation between privilege levels. Pointer validation consists of the following checks:

1. Checking access rights to determine if the segment type is compatible with its use.
2. Checking read/write rights.
3. Checking if the pointer offset exceeds the segment limit.
4. Checking if the supplier of the pointer is allowed to access the segment.
5. Checking the offset alignment.

The processor automatically performs first, second, and third checks during instruction execution. Software must explicitly request the fourth check by issuing an ARPL instruction. The fifth check (offset alignment) is performed automatically at privilege level 3 if alignment checking is turned on. Offset alignment does not affect isolation of privilege levels.

6.10.1 Checking Access Rights (LAR Instruction)

When the processor accesses a segment using a far pointer, it performs an access rights check on the segment descriptor pointed to by the far pointer. This check is performed to determine if type and privilege level (DPL) of the segment descriptor are compatible with the operation to be performed. For example, when making a far call in protected mode, the segment-descriptor type must be for a conforming or nonconforming code segment, a call

gate, a task gate, or a TSS. Then, if the call is to a nonconforming code segment, the DPL of the code segment must be equal to the CPL, and the RPL of the code segment's segment selector must be less than or equal to the DPL. If type or privilege level are found to be incompatible, the appropriate exception is generated.

To prevent type incompatibility exceptions from being generated, software can check the access rights of a segment descriptor using the LAR (load access rights) instruction. The LAR instruction specifies the segment selector for the segment descriptor whose access rights are to be checked and a destination register. The instruction then performs the following operations:

1. Check that the segment selector is not null.
2. Checks that the segment selector points to a segment descriptor that is within the descriptor table limit (GDT or LDT).
3. Checks that the segment descriptor is a code, data, LDT, call gate, task gate, or TSS segment-descriptor type.
4. If the segment is not a conforming code segment, checks if the segment descriptor is visible at the CPL (that is, if the CPL and the RPL of the segment selector are less than or equal to the DPL).
5. If the privilege level and type checks pass, loads the second doubleword of the segment descriptor into the destination register (masked by the value 00FxFF00H, where X indicates that the corresponding 4 bits are undefined) and sets the ZF flag in the EFLAGS register. If the segment selector is not visible at the current privilege level or is an invalid type for the LAR instruction, the instruction does not modify the destination register and clears the ZF flag.

Once loaded in the destination register, software can perform additional checks on the access rights information.

6.10.2 Checking Read/Write Rights (VERR and VERW Instructions)

When the processor accesses any code or data segment it checks the read/write privileges assigned to the segment to verify that the intended read or write operation is allowed. Software can check read/write rights using the VERR (verify for reading) and VERW (verify for writing) instructions. Both these instructions specify the segment selector for the segment being checked. The instructions then perform the following operations:

1. Check that the segment selector is not null.
2. Checks that the segment selector points to a segment descriptor that is within the descriptor table limit (GDT or LDT).
3. Checks that the segment descriptor is a code or data-segment descriptor type.
4. If the segment is not a conforming code segment, checks if the segment descriptor is visible at the CPL (that is, if the CPL and the RPL of the segment selector are less than or equal to the DPL).
5. Checks that the segment is readable (for the VERR instruction) or writable (for the VERW) instruction.

The VERR instruction sets the ZF flag in the EFLAGS register if the segment is visible at the CPL and readable; the VERW sets the ZF flag if the segment is visible and writable. (Code segments are never writable.) The ZF flag is cleared if any of these checks fail.

6.10.3 Checking That the Pointer Offset Is Within Limits (LSL Instruction)

When the processor accesses any segment it performs a limit check to ensure that the offset is within the limit of the segment. Software can perform this limit check using the LSL (load segment limit) instruction. Like the LAR instruction, the LSL instruction specifies the segment selector for the segment descriptor whose limit is to be checked and a destination register. The instruction then performs the following operations:

1. Check that the segment selector is not null.
2. Checks that the segment selector points to a segment descriptor that is within the descriptor table limit (GDT or LDT).
3. Checks that the segment descriptor is a code, data, LDT, or TSS segment-descriptor type.
4. If the segment is not a conforming code segment, checks if the segment descriptor is visible at the CPL (that is, if the CPL and the RPL of the segment selector less than or equal to the DPL).

5. If the privilege level and type checks pass, loads the unscrambled limit (the limit scaled according to the setting of the G flag in the segment descriptor) into the destination register and sets the ZF flag in the EFLAGS register. If the segment selector is not visible at the current privilege level or is an invalid type for the LSL instruction, the instruction does not modify the destination register and clears the ZF flag.

Once loaded in the destination register, software can compare the segment limit with the offset of a pointer.

6.10.4 Checking Caller Access Privileges (ARPL Instruction)

The requestor's privilege level (RPL) field of a segment selector is intended to carry the privilege level of a calling procedure (the calling procedure's CPL) to a called procedure. The called procedure then uses the RPL to determine if access to a segment is allowed. The RPL is said to "weaken" the privilege level of the called procedure to that of the RPL.

Operating-system procedures typically use the RPL to prevent less privileged application programs from accessing data located in more privileged segments. When an operating-system procedure (the called procedure) receives a segment selector from an application program (the calling procedure), it sets the segment selector's RPL to the privilege level of the calling procedure. Then, when the operating system uses the segment selector to access its associated segment, the processor performs privilege checks using the calling procedure's privilege level (stored in the RPL) rather than the numerically lower privilege level (the CPL) of the operating-system procedure. The RPL thus ensures that the operating system does not access a segment on behalf of an application program unless that program itself has access to the segment.

Figure 6-15 shows an example of how the processor uses the RPL field. In this example, an application program (located in code segment A) possesses a segment selector (segment selector D1) that points to a privileged data structure (that is, a data structure located in a data segment D at privilege level 0).

The application program cannot access data segment D, because it does not have sufficient privilege, but the operating system (located in code segment C) can. So, in an attempt to access data segment D, the application program executes a call to the operating system and passes segment selector D1 to the operating system as a parameter on the stack. Before passing the segment selector, the (well behaved) application program sets the RPL of the segment selector to its current privilege level (which in this example is 3). If the operating system attempts to access data segment D using segment selector D1, the processor compares the CPL (which is now 0 following the call), the RPL of segment selector D1, and the DPL of data segment D (which is 0). Since the RPL is greater than the DPL, access to data segment D is denied. The processor's protection mechanism thus protects data segment D from access by the operating system, because application program's privilege level (represented by the RPL of segment selector B) is greater than the DPL of data segment D.

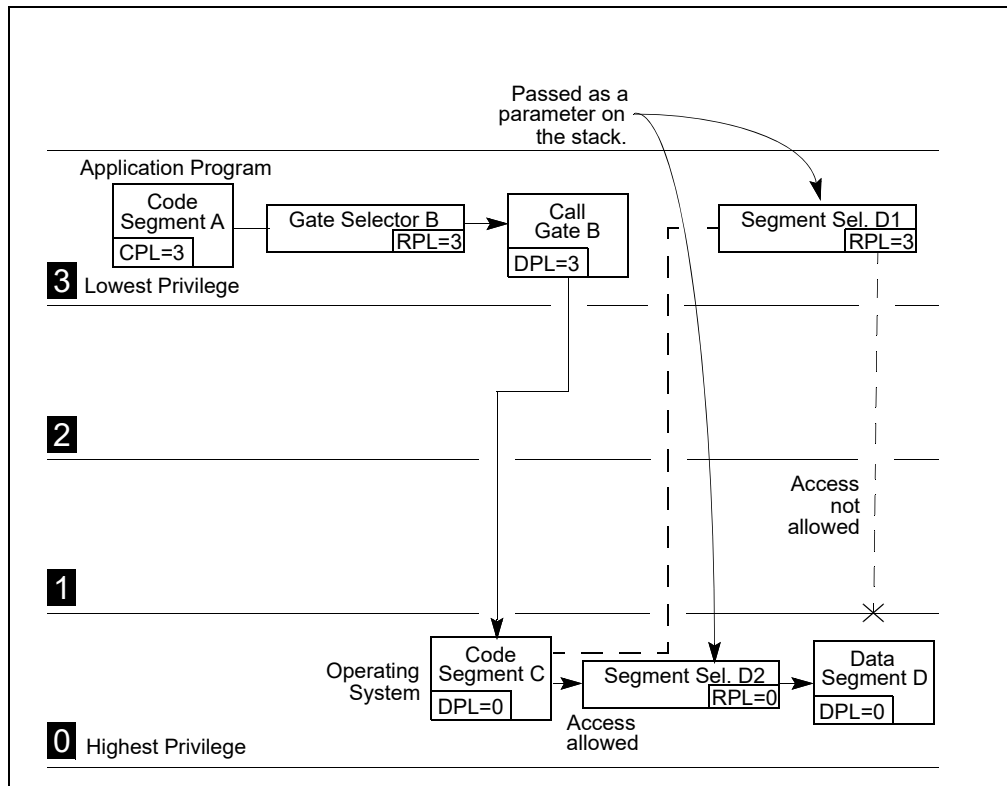


Figure 6-15. Use of RPL to Weaken Privilege Level of Called Procedure

Now assume that instead of setting the RPL of the segment selector to 3, the application program sets the RPL to 0 (segment selector D2). The operating system can now access data segment D, because its CPL and the RPL of segment selector D2 are both equal to the DPL of data segment D.

Because the application program is able to change the RPL of a segment selector to any value, it can potentially use a procedure operating at a numerically lower privilege level to access a protected data structure. This ability to lower the RPL of a segment selector breaches the processor's protection mechanism.

Because a called procedure cannot rely on the calling procedure to set the RPL correctly, operating-system procedures (executing at numerically lower privilege-levels) that receive segment selectors from numerically higher privilege-level procedures need to test the RPL of the segment selector to determine if it is at the appropriate level. The ARPL (adjust requested privilege level) instruction is provided for this purpose. This instruction adjusts the RPL of one segment selector to match that of another segment selector.

The example in Figure 6-15 demonstrates how the ARPL instruction is intended to be used. When the operating-system receives segment selector D2 from the application program, it uses the ARPL instruction to compare the RPL of the segment selector with the privilege level of the application program (represented by the code-segment selector pushed onto the stack). If the RPL is less than application program's privilege level, the ARPL instruction changes the RPL of the segment selector to match the privilege level of the application program (segment selector D1). Using this instruction thus prevents a procedure running at a numerically higher privilege level from accessing numerically lower privilege-level (more privileged) segments by lowering the RPL of a segment selector.

Note that the privilege level of the application program can be determined by reading the RPL field of the segment selector for the application-program's code segment. This segment selector is stored on the stack as part of the call to the operating system. The operating system can copy the segment selector from the stack into a register for use as an operand for the ARPL instruction.

6.10.5 Checking Alignment

When the CPL is 3, alignment of memory references can be checked by setting the AM flag in the CR0 register and the AC flag in the EFLAGS register. Unaligned memory references generate alignment exceptions (#AC). The processor does not generate alignment exceptions when operating at privilege level 0, 1, or 2. See Table 7-7 for a description of the alignment requirements when alignment checking is enabled.

6.11 PAGE-LEVEL PROTECTION

Page-level protection can be used alone or applied to segments. When page-level protection is used with the flat memory model, it allows supervisor code and data (the operating system or executive) to be protected from user code and data (application programs). It also allows pages containing code to be write protected. When the segment- and page-level protection are combined, page-level read/write protection allows more protection granularity within segments.

With page-level protection (as with segment-level protection) each memory reference is checked to verify that protection checks are satisfied. All checks are made before the memory cycle is started, and any violation prevents the cycle from starting and results in a page-fault exception being generated. Because checks are performed in parallel with address translation, there is no performance penalty.

The processor performs two page-level protection checks:

- Restriction of addressable domain (supervisor and user modes).
- Page type (read only or read/write).

Violations of either of these checks results in a page-fault exception being generated. See Chapter 7, "Interrupt 14—Page-Fault Exception (#PF)," for an explanation of the page-fault exception mechanism. This chapter describes the protection violations which lead to page-fault exceptions.

6.11.1 Page-Protection Flags

Protection information for pages is contained in two flags in a paging-structure entry (see Chapter 5): the read/write flag (bit 1) and the user/supervisor flag (bit 2). The protection checks use the flags in all paging structures.

6.11.2 Restricting Addressable Domain

The page-level protection mechanism allows restricting access to pages based on two privilege levels:

- Supervisor mode (U/S flag is 0)—(Most privileged) For the operating system or executive, other system software (such as device drivers), and protected system data (such as page tables).
- User mode (U/S flag is 1)—(Least privileged) For application code and data.

The segment privilege levels map to the page privilege levels as follows. If the processor is currently operating at a CPL of 0, 1, or 2, it is in supervisor mode; if it is operating at a CPL of 3, it is in user mode. When the processor is in supervisor mode, it can access all pages; when in user mode, it can access only user-level pages. (Note that the WP flag in control register CR0 modifies the supervisor permissions, as described in Section 6.11.3, "Page Type.")

Note that to use the page-level protection mechanism, code and data segments must be set up for at least two segment-based privilege levels: level 0 for supervisor code and data segments and level 3 for user code and data segments. (In this model, the stacks are placed in the data segments.) To minimize the use of segments, a flat memory model can be used (see Section 3.2.1, "Basic Flat Model").

Here, the user and supervisor code and data segments all begin at address zero in the linear address space and overlay each other. With this arrangement, operating-system code (running at the supervisor level) and application code (running at the user level) can execute as if there are no segments. Protection between operating-system and application code and data is provided by the processor's page-level protection mechanism.

6.11.3 Page Type

The page-level protection mechanism recognizes two page types:

- Read-only access (R/W flag is 0).
- Read/write access (R/W flag is 1).

When the processor is in supervisor mode and the WP flag in register CR0 is clear (its state following reset initialization), all pages are both readable and writable (write-protection is ignored). When the processor is in user mode, it can write only to user-mode pages that are read/write accessible. User-mode pages which are read/write or read-only are readable; supervisor-mode pages are neither readable nor writable from user mode. A page-fault exception is generated on any attempt to violate the protection rules.

Starting with the P6 family, Intel processors allow user-mode pages to be write-protected against supervisor-mode access. Setting CR0.WP = 1 enables supervisor-mode sensitivity to write protected pages. If CR0.WP = 1, read-only pages are not writable from any privilege level. This supervisor write-protect feature is useful for implementing a “copy-on-write” strategy used by some operating systems, such as UNIX*, for task creation (also called forking or spawning). When a new task is created, it is possible to copy the entire address space of the parent task. This gives the child task a complete, duplicate set of the parent's segments and pages. An alternative copy-on-write strategy saves memory space and time by mapping the child's segments and pages to the same segments and pages used by the parent task. A private copy of a page gets created only when one of the tasks writes to the page. By using the WP flag and marking the shared pages as read-only, the supervisor can detect an attempt to write to a page, and can copy the page at that time.

6.11.4 Combining Protection of Both Levels of Page Tables

For any one page, the protection attributes of its page-directory entry (first-level page table) may differ from those of its page-table entry (second-level page table). The processor checks the protection for a page in both its page-directory and the page-table entries. Table 6-3 shows the protection provided by the possible combinations of protection attributes when the WP flag is clear.

6.11.5 Overrides to Page Protection

The following types of memory accesses are checked as if they are privilege-level 0 accesses, regardless of the CPL at which the processor is currently operating:

- Access to segment descriptors in the GDT, LDT, or IDT.
- Access to an inner-privilege-level stack during an inter-privilege-level call or a call to an exception or interrupt handler, when a change of privilege level occurs.

6.12 COMBINING PAGE AND SEGMENT PROTECTION

When paging is enabled, the processor evaluates segment protection first, then evaluates page protection. If the processor detects a protection violation at either the segment level or the page level, the memory access is not carried out and an exception is generated. If an exception is generated by segmentation, no paging exception is generated.

Page-level protections cannot be used to override segment-level protection. For example, a code segment is by definition not writable. If a code segment is paged, setting the R/W flag for the pages to read-write does not make the pages writable. Attempts to write into the pages will be blocked by segment-level protection checks.

Page-level protection can be used to enhance segment-level protection. For example, if a large read-write data segment is paged, the page-protection mechanism can be used to write-protect individual pages.

Table 6-3. Combined Page-Directory and Page-Table Protection

Page-Directory Entry		Page-Table Entry		Combined Effect	
Privilege	Access Type	Privilege	Access Type	Privilege	Access Type

Table 6-3. Combined Page-Directory and Page-Table Protection

User	Read-Only	User	Read-Only	User	Read-Only
User	Read-Only	User	Read-Write	User	Read-Only
User	Read-Write	User	Read-Only	User	Read-Only
User	Read-Write	User	Read-Write	User	Read/Write
User	Read-Only	Supervisor	Read-Only	Supervisor	Read/Write*
User	Read-Only	Supervisor	Read-Write	Supervisor	Read/Write*
User	Read-Write	Supervisor	Read-Only	Supervisor	Read/Write*
User	Read-Write	Supervisor	Read-Write	Supervisor	Read/Write
Supervisor	Read-Only	User	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Only	User	Read-Write	Supervisor	Read/Write*
Supervisor	Read-Write	User	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Write	User	Read-Write	Supervisor	Read/Write
Supervisor	Read-Only	Supervisor	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Only	Supervisor	Read-Write	Supervisor	Read/Write*
Supervisor	Read-Write	Supervisor	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Write	Supervisor	Read-Write	Supervisor	Read/Write

NOTE:

* If CRO.WP = 1, access type is determined by the R/W flags of the page-directory and page-table entries. If CRO.WP = 0, supervisor privilege permits read-write access.

6.13 PAGE-LEVEL PROTECTION AND EXECUTE-DISABLE BIT

In addition to page-level protection offered by the U/S and R/W flags, paging structures used with PAE paging, 4-level paging,¹ and 5-level paging provide the execute-disable bit (see Chapter 5, “Paging”). This bit offers additional protection for data pages.

An Intel 64 or IA-32 processor with the execute-disable bit capability can prevent data pages from being used by malicious software to execute code. This capability is provided in:

- 32-bit protected mode with PAE enabled.
- IA-32e mode.

While the execute-disable bit capability does not introduce new instructions, it does require operating systems to use a PAE-enabled environment and establish a page-granular protection policy for memory pages.

If the execute-disable bit of a memory page is set, that page can be used only as data. An attempt to execute code from a memory page with the execute-disable bit set causes a page-fault exception.

The execute-disable capability is not supported with 32-bit paging. Existing page-level protection mechanisms (see Section 6.11, “Page-Level Protection”) continue to apply to memory pages independent of the execute-disable setting.

6.13.1 Detecting and Enabling the Execute-Disable Capability

Software can detect the presence of the execute-disable capability using the CPUID instruction. CPUID.80000001H:EDX.NX[20] = 1 indicates the capability is available.

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

If the capability is available, software can enable it by setting IA32_EFER.NXE[bit 11] to 1. IA32_EFER is available if CPUID.80000001H:EDX[20 or 29] = 1.

If the execute-disable capability is not available, a write to set IA32_EFER.NXE produces a #GP exception. See Table 6-4.

Table 6-4. Extended Feature Enable MSR (IA32_EFER)

63:12	11	10	9	8	7:1	0
Reserved	Execute-disable bit enable (NXE)	IA-32e mode active (LMA)	Reserved	IA-32e mode enable (LME)	Reserved	SysCall enable (SCE)

6.13.2 Execute-Disable Page Protection

The execute-disable bit in the paging structures enhances page protection for data pages. Instructions cannot be fetched from a memory page if IA32_EFER.NXE = 1 and the execute-disable bit is set in any of the paging-structure entries used to map the page. Table 6-5 lists the valid usage of a page in relation to the value of execute-disable bit (bit 63) of the corresponding entry in each level of the paging structures. Execute-disable protection can be activated using the execute-disable bit at any level of the paging structure, irrespective of the corresponding entry in other levels. When execute-disable protection is not activated, the page can be used as code or data.

Table 6-5. Page Level Protection Matrix with Execute-Disable Bit Capability with 4-Level Paging

Execute Disable Bit Value (Bit 63)				Valid Usage
PML4	PDP	PDE	PTE	
Bit 63 = 1	*	*	*	Data
*	Bit 63 = 1	*	*	Data
*	*	Bit 63 = 1	*	Data
*	*	*	Bit 63 = 1	Data
Bit 63 = 0	Bit 63 = 0	Bit 63 = 0	Bit 63 = 0	Data/Code

NOTES:

* Value not checked.

In legacy PAE-enabled mode, Table 6-6 and Table 6-7 show the effect of setting the execute-disable bit for code and data pages.

Table 6-6. 4-KByte Page Level Protection Matrix with Execute-Disable Bit Capability with PAE Paging

Execute Disable Bit Value (Bit 63)		Valid Usage
PDE	PTE	
Bit 63 = 1	*	Data
*	Bit 63 = 1	Data
Bit 63 = 0	Bit 63 = 0	Data/Code

NOTE:

* Value not checked.

Table 6-7. 2-MByte Page Level Protection with Execute-Disable Bit Capability with PAE Paging

Execute Disable Bit Value (Bit 63)	Valid Usage
PDE	
Bit 63 = 1	Data
Bit 63 = 0	Data/Code

6.13.3 Reserved Bit Checking

The processor enforces reserved bit checking in paging data structure entries. The bits being checked varies with paging mode and may vary with the size of physical address space.

Table 6-8 shows the reserved bits that are checked when the execute disable bit capability is enabled (CR4.PAE = 1 and IA32_EFER.NXE = 1). Table 6-8 and Table 6-9 show the following paging modes:

- Non-PAE 4-KByte paging: 4-KByte-page only paging (CR4.PAE = 0, CR4.PSE = 0).
- PSE36: 4-KByte and 4-MByte pages (CR4.PAE = 0, CR4.PSE = 1).
- PAE: 4-KByte and 2-MByte pages (CR4.PAE = 1).

Table 6-8. Page Level Protection Matrix with Execute-Disable Bit Capability Enabled

Mode	Paging Mode	Check Bits
32-bit	4-KByte paging (non-PAE)	No reserved bits checked
	PSE36 - PDE, 4-MByte page	Bit 21
	PSE36 - PDE, 4-KByte page	No reserved bits checked
	PSE36 - PTE	No reserved bits checked
	PAE - PDP table entry	Bits 63:MAXPHYADDR, 8:5, and 2:1
	PAE - PDE, 2-MByte page	Bits 62:MAXPHYADDR and 20:13
	PAE - PDE, 4-KByte page	Bits 62:MAXPHYADDR
	PAE - PTE	Bits 62:MAXPHYADDR
64-bit	PML5E	Bits 51:MAXPHYADDR
	PML4E	Bits 51:MAXPHYADDR
	PDPTE	Bits 51:MAXPHYADDR
	PDE, 2-MByte page	Bits 51:MAXPHYADDR and 20:13
	PDE, 4-KByte page	Bits 51:MAXPHYADDR
	PTE	Bits 51:MAXPHYADDR

The reserved bit checking depends on the physical-address width supported by the processor. This width is denoted MAXPHYADDR and is enumerated in CPUID.80000008H:EAX[7:0].¹

If execute disable bit capability is not enabled or not available, reserved bit checking in 64-bit mode includes bit 63 and additional bits. This and reserved bit checking for legacy 32-bit paging modes are shown in Table 6-9.

Table 6-9. Reserved Bit Checking with Execute-Disable Bit Capability Not Enabled

Mode	Paging Mode	Check Bits
32-bit	KByte paging (non-PAE)	No reserved bits checked
	PSE36 - PDE, 4-MByte page	Bit 21
	PSE36 - PDE, 4-KByte page	No reserved bits checked
	PSE36 - PTE	No reserved bits checked
	PAE - PDP table entry	Bits 63:MAXPHYADDR, 8:5, and 2:1
	PAE - PDE, 2-MByte page	Bits 63:MAXPHYADDR and 20:13
	PAE - PDE, 4-KByte page	Bits 63:MAXPHYADDR
	PAE - PTE	Bits 63:MAXPHYADDR
64-bit	PML5E	Bit 63, bits 51:MAXPHYADDR
	PML4E	Bit 63, bits 51:MAXPHYADDR
	PDPTE	Bit 63, bits 51:MAXPHYADDR
	PDE, 2-MByte page	Bit 63, bits 51:MAXPHYADDR and 20:13
	PDE, 4-KByte page	Bit 63, bits 51:MAXPHYADDR
	PTE	Bit 63, bits 51:MAXPHYADDR

1. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.

6.13.4 Exception Handling

When execute disable bit capability is enabled (IA32_EFER.NXE = 1), conditions for a page fault to occur include the same conditions that apply to an Intel 64 or IA-32 processor without execute disable bit capability plus the following new condition: an instruction fetch to a linear address that translates to physical address in a memory page that has the execute-disable bit set.

An Execute Disable Bit page fault can occur at all privilege levels. It can occur on any instruction fetch, including (but not limited to): near branches, far branches, CALL/RET/INT/IRET execution, sequential instruction fetches, and task switches. The execute-disable bit in the page translation mechanism is checked only when:

- IA32_EFER.NXE = 1.
- The instruction translation look-aside buffer (ITLB) is loaded with a page that is not already present in the ITLB.

13. Updates to Chapter 7, Volume 3A

Change bars and text show changes to Chapter 7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- Added behavior for FRED transitions in Section 7.2, "Exception and Interrupt Vectors".
- Documented behavior when FRED transitions are enabled in Section 7.7.1, "Handling Multiple NMIs".
- Updated Section 7.8.1, "Masking Maskable Hardware Interrupts" for FRED.
- Noted that FRED event delivery is an alternative to IDT in Section 7.10, "Interrupt Descriptor Table (IDT)".
- Added notes in Section 7.11, "IDT Descriptors", Section 7.12, "Exception and Interrupt Handling", and Section 7.14, "Exception and Interrupt Handling in 64-bit Mode" that clarify these sections are not applicable when FRED transitions are enabled.
- In Section 7.13, "Error Code", documented that IDT bit of the error code is not set when FRED transitions are enabled.
- Corrected name of MSR IA32_INTERRUPT_SSP_TABLE_ADDR in Section 7.14.5 and Figure 7-10.

CHAPTER 7

INTERRUPT AND EXCEPTION HANDLING

This chapter describes the interrupt and exception-handling mechanism when operating in protected mode on an Intel 64 or IA-32 processor. Most of the information provided here also applies to interrupt and exception mechanisms used in real-address, virtual-8086 mode, and 64-bit mode.

Chapter 23, “8086 Emulation,” describes information specific to interrupt and exception mechanisms in real-address and virtual-8086 mode. Section 7.14, “Exception and Interrupt Handling in 64-bit Mode,” describes information specific to interrupt and exception mechanisms in IA-32e mode and 64-bit sub-mode.

7.1 INTERRUPT AND EXCEPTION OVERVIEW

Interrupts and exceptions are events that indicate that a condition exists somewhere in the system, the processor, or within the currently executing program or task that requires the attention of a processor. They typically result in a forced transfer of execution from the currently running program or task to a special software routine or task called an interrupt handler or an exception handler. The action taken by a processor in response to an interrupt or exception is referred to as servicing or handling the interrupt or exception.

Interrupts occur at random times during the execution of a program, in response to signals from hardware. System hardware uses interrupts to handle events external to the processor, such as requests to service peripheral devices. Software can also generate interrupts by executing the `INT n` instruction.

Exceptions occur when the processor detects an error condition while executing an instruction, such as division by zero. The processor detects a variety of error conditions including protection violations, page faults, and internal machine faults. The machine-check architecture of the Pentium 4, Intel Xeon, P6 family, and Pentium processors also permits a machine-check exception to be generated when internal hardware errors and bus errors are detected.

When an interrupt is received or an exception is detected, the currently running procedure or task is suspended while the processor executes an interrupt or exception handler. When execution of the handler is complete, the processor resumes execution of the interrupted procedure or task. The resumption of the interrupted procedure or task happens without loss of program continuity, unless recovery from an exception was not possible or an interrupt caused the currently running program to be terminated.

This chapter describes the processor’s interrupt and exception-handling mechanism, when operating in protected mode. A description of the exceptions and the conditions that cause them to be generated is given at the end of this chapter.

7.2 EXCEPTION AND INTERRUPT VECTORS

To aid in handling exceptions and interrupts, each architecturally defined exception and each interrupt condition requiring special handling by the processor is assigned a unique identification number, called a vector number. The processor uses the vector number assigned to an exception or interrupt as an index into the interrupt descriptor table (IDT). The table provides the entry point to an exception or interrupt handler (see Section 7.10, “Interrupt Descriptor Table (IDT)”). *When FRED transitions are enabled, the vector number does not reference the IDT but is instead saved on the stack by FRED event delivery; see Chapter 8, “Flexible Return and Event Delivery (FRED).”*

The allowable range for vector numbers is 0 to 255. Vector numbers in the range 0 through 31 are reserved by the Intel 64 and IA-32 architectures for architecture-defined exceptions and interrupts. Not all of the vector numbers in this range have a currently defined function. The unassigned vector numbers in this range are reserved. Do not use the reserved vector numbers.

Vector numbers in the range 32 to 255 are designated as user-defined interrupts and are not reserved by the Intel 64 and IA-32 architecture. These interrupts are generally assigned to external I/O devices to enable those devices to send interrupts to the processor through one of the external hardware interrupt mechanisms (see Section 7.3, “Sources of Interrupts”).

Table 7-1 shows vector number assignments for architecturally defined exceptions and for the NMI interrupt. This table gives the exception type (see Section 7.5, “Exception Classifications”) and indicates whether an error code is saved on the stack for the exception. The source of each predefined exception and the NMI interrupt is also given.

7.3 SOURCES OF INTERRUPTS

The processor receives interrupts from two sources:

- External (hardware generated) interrupts.
- Software-generated interrupts.

7.3.1 External Interrupts

External interrupts are received through pins on the processor or through the local APIC. The primary interrupt pins on Pentium 4, Intel Xeon, P6 family, and Pentium processors are the LINT[1:0] pins, which are connected to the local APIC (see Chapter 13, “Advanced Programmable Interrupt Controller (APIC)”). When the local APIC is enabled, the LINT[1:0] pins can be programmed through the APIC’s local vector table (LVT) to be associated with any of the processor’s exception or interrupt vectors.

When the local APIC is global/hardware disabled, these pins are configured as INTR and NMI pins, respectively. Asserting the INTR pin signals the processor that an external interrupt has occurred. The processor reads from the system bus the interrupt vector number provided by an external interrupt controller, such as an 8259A (see Section 7.2, “Exception and Interrupt Vectors”). Asserting the NMI pin signals a non-maskable interrupt (NMI), which is assigned to interrupt vector 2.

Table 7-1. Protected-Mode Exceptions and Interrupts

Vector	Mnemonic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	Debug Exception	Fault/ Trap	No	Instruction, data, and I/O breakpoints; single-step; and others.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD instruction or reserved opcode.
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction. ¹
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.
12	#SS	Stack-Segment Fault	Fault	Yes	Stack operations and SS register loads.
13	#GP	General Protection	Fault	Yes	Any memory reference and other protection checks.
14	#PF	Page Fault	Fault	Yes	Any memory reference.

Table 7-1. Protected-Mode Exceptions and Interrupts (Contd.)

Vector	Mnemonic	Description	Type	Error Code	Source
15	—	(Intel reserved. Do not use.)		No	
16	#MF	x87 FPU Floating-Point Error (Math Fault)	Fault	No	x87 FPU floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Fault	Yes	Any data reference in memory. ²
18	#MC	Machine Check	Abort	No	Error codes (if any) and source are model dependent. ³
19	#XM	SIMD Floating-Point Exception	Fault	No	SSE/SSE2/SSE3 floating-point instructions ⁴
20	#VE	Virtualization Exception	Fault	No	EPT violations ⁵
21	#CP	Control Protection Exception	Fault	Yes	RET, IRET, RSTORSSP, and SETSSBSY instructions can generate this exception. When CET indirect branch tracking is enabled, this exception can be generated due to a missing ENDBRANCH instruction at target of an indirect call or jump.
22-31	—	Intel reserved. Do not use.			
32-255	—	User Defined (Non-reserved) Interrupts	Interrupt		External interrupt or INT <i>n</i> instruction.

NOTES:

- Processors after the Intel386 processor do not generate this exception.
- This exception was introduced in the Intel486 processor.
- This exception was introduced in the Pentium processor and enhanced in the P6 family processors.
- This exception was introduced in the Pentium III processor.
- This exception can occur only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.

The processor’s local APIC is normally connected to a system-based I/O APIC. Here, external interrupts received at the I/O APIC’s pins can be directed to the local APIC through the system bus (Pentium 4, Intel Core Duo, Intel Core 2, Intel Atom, and Intel Xeon processors) or the APIC serial bus (P6 family and Pentium processors). The I/O APIC determines the vector number of the interrupt and sends this number to the local APIC. When a system contains multiple processors, processors can also send interrupts to one another by means of the system bus (Pentium 4, Intel Core Duo, Intel Core 2, Intel Atom, and Intel Xeon processors) or the APIC serial bus (P6 family and Pentium processors).

The LINT[1:0] pins are not available on the Intel486 processor and earlier Pentium processors that do not contain an on-chip local APIC. These processors have dedicated NMI and INTR pins. With these processors, external interrupts are typically generated by a system-based interrupt controller (8259A), with the interrupts being signaled through the INTR pin.

Note that several other pins on the processor can cause a processor interrupt to occur. However, these interrupts are not handled by the interrupt and exception mechanism described in this chapter. These pins include the RESET#, FLUSH#, STPCLK#, SMI#, R/S#, and INIT# pins. Whether they are included on a particular processor is implementation dependent. Pin functions are described in the data books for the individual processors. The SMI# pin is described in Chapter 34, “System Management Mode.”

7.3.2 Maskable Hardware Interrupts

Any external interrupt that is delivered to the processor by means of the INTR pin or through the local APIC is called a maskable hardware interrupt. Maskable hardware interrupts that can be delivered through the INTR pin include all IA-32 architecture defined interrupt vectors from 0 through 255; those that can be delivered through the local APIC include interrupt vectors 16 through 255.

The IF flag in the EFLAGS register permits all maskable hardware interrupts to be masked as a group (see Section 7.8.1, “Masking Maskable Hardware Interrupts”). Note that when interrupts 0 through 15 are delivered through the local APIC, the APIC indicates the receipt of an illegal vector.

7.3.3 Software-Generated Interrupts

The `INT n` instruction permits interrupts to be generated from within software by supplying an interrupt vector number as an operand. For example, the `INT 35` instruction forces an implicit call to the interrupt handler for interrupt 35.

Any of the interrupt vectors from 0 to 255 can be used as a parameter in this instruction. If the processor’s predefined NMI vector is used, however, the response of the processor will not be the same as it would be from an NMI interrupt generated in the normal manner. If vector number 2 (the NMI vector) is used in this instruction, the NMI interrupt handler is called, but the processor’s NMI-handling hardware is not activated.

Interrupts generated in software with the `INT n` instruction cannot be masked by the IF flag in the EFLAGS register.

7.4 SOURCES OF EXCEPTIONS

The processor receives exceptions from three sources:

- Processor-detected program-error exceptions.
- Software-generated exceptions.
- Machine-check exceptions.

7.4.1 Program-Error Exceptions

The processor generates one or more exceptions when it detects program errors during the execution in an application program or the operating system or executive. Intel 64 and IA-32 architectures define a vector number for each processor-detectable exception. Exceptions are classified as **faults**, **traps**, and **aborts** (see Section 7.5, “Exception Classifications”).

7.4.2 Software-Generated Exceptions

The `INTO`, `INT1`, `INT3`, and `BOUND` instructions permit exceptions to be generated in software. These instructions allow checks for exception conditions to be performed at points in the instruction stream. For example, `INT3` causes a breakpoint exception to be generated.

The `INT n` instruction can be used to emulate exceptions in software; but there is a limitation.¹ If `INT n` provides a vector for one of the architecturally-defined exceptions, the processor generates an interrupt to the correct vector (to access the exception handler) but does not push an error code on the stack. This is true even if the associated hardware-generated exception normally produces an error code. The exception handler will still attempt to pop an error code from the stack while handling the exception. Because no error code was pushed, the handler will pop off and discard the EIP instead (in place of the missing error code). This sends the return to the wrong location.

7.4.3 Machine-Check Exceptions

The P6 family and Pentium processors provide both internal and external machine-check mechanisms for checking the operation of the internal chip hardware and bus transactions. These mechanisms are implementation dependent. When a machine-check error is detected, the processor signals a machine-check exception (vector 18) and returns an error code.

1. The `INT n` instruction has opcode `CD` following by an immediate byte encoding the value of *n*. In contrast, `INT1` has opcode `F1` and `INT3` has opcode `CC`.

See Chapter 7, “Interrupt 18—Machine-Check Exception (#MC),” and Chapter 18, “Machine-Check Architecture,” for more information about the machine-check mechanism.

7.5 EXCEPTION CLASSIFICATIONS

Exceptions are classified as **faults**, **traps**, or **aborts** depending on the way they are reported and whether the instruction that caused the exception can be restarted without loss of program or task continuity.

- **Faults** — A fault is an exception that can generally be corrected and that, once corrected, allows the program to be restarted with no loss of continuity. When a fault is reported, the processor restores the machine state to the state prior to the beginning of execution of the faulting instruction. The return address (saved contents of the CS and EIP registers) for the fault handler points to the faulting instruction, rather than to the instruction following the faulting instruction.
- **Traps** — A trap is an exception that is reported immediately following the execution of the trapping instruction. Traps allow execution of a program or task to be continued without loss of program continuity. The return address for the trap handler points to the instruction to be executed after the trapping instruction.
- **Aborts** — An abort is an exception that does not always report the precise location of the instruction causing the exception and does not allow a restart of the program or task that caused the exception. Aborts are used to report severe errors, such as hardware errors and inconsistent or illegal values in system tables.

NOTE

One exception subset normally reported as a fault is not restartable. Such exceptions result in loss of some processor state. For example, executing a POPAD instruction where the stack frame crosses over the end of the stack segment causes a fault to be reported. In this situation, the exception handler sees that the instruction pointer (CS:EIP) has been restored as if the POPAD instruction had not been executed. However, internal processor state (the general-purpose registers) will have been modified. Such cases are considered programming errors. An application causing this class of exceptions should be terminated by the operating system.

7.6 PROGRAM OR TASK RESTART

To allow the restarting of program or task following the handling of an exception or an interrupt, all exceptions (except aborts) are guaranteed to report exceptions on an instruction boundary. All interrupts are guaranteed to be taken on an instruction boundary.

For fault-class exceptions, the return instruction pointer (saved when the processor generates an exception) points to the faulting instruction. So, when a program or task is restarted following the handling of a fault, the faulting instruction is restarted (re-executed). Restarting the faulting instruction is commonly used to handle exceptions that are generated when access to an operand is blocked. The most common example of this type of fault is a page-fault exception (#PF) that occurs when a program or task references an operand located on a page that is not in memory. When a page-fault exception occurs, the exception handler can load the page into memory and resume execution of the program or task by restarting the faulting instruction. To ensure that the restart is handled transparently to the currently executing program or task, the processor saves the necessary registers and stack pointers to allow a restart to the state prior to the execution of the faulting instruction.

For trap-class exceptions, the return instruction pointer points to the instruction following the trapping instruction. If a trap is detected during an instruction which transfers execution, the return instruction pointer reflects the transfer. For example, if a trap is detected while executing a JMP instruction, the return instruction pointer points to the destination of the JMP instruction, not to the next address past the JMP instruction. All trap exceptions allow program or task restart with no loss of continuity. For example, the overflow exception is a trap exception. Here, the return instruction pointer points to the instruction following the INTO instruction that tested EFLAGS.OF (overflow) flag. The trap handler for this exception resolves the overflow condition. Upon return from the trap handler, program or task execution continues at the instruction following the INTO instruction.

The abort-class exceptions do not support reliable restarting of the program or task. Abort handlers are designed to collect diagnostic information about the state of the processor when the abort exception occurred and then shut down the application and system as gracefully as possible.

Interrupts rigorously support restarting of interrupted programs and tasks without loss of continuity. The return instruction pointer saved for an interrupt points to the next instruction to be executed at the instruction boundary where the processor took the interrupt. If the instruction just executed has a repeat prefix, the interrupt is taken at the end of the current iteration with the registers set to execute the next iteration.

The ability of a P6 family processor to speculatively execute instructions does not affect the taking of interrupts by the processor. Interrupts are taken at instruction boundaries located during the retirement phase of instruction execution; so they are always taken in the “in-order” instruction stream. See Chapter 2, “Intel® 64 and IA-32 Architectures,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for more information about the P6 family processors’ microarchitecture and its support for out-of-order instruction execution.

Note that the Pentium processor and earlier IA-32 processors also perform varying amounts of prefetching and preliminary decoding. With these processors as well, exceptions and interrupts are not signaled until actual “in-order” execution of the instructions. For a given code sample, the signaling of exceptions occurs uniformly when the code is executed on any family of IA-32 processors (except where new exceptions or new opcodes have been defined).

7.7 NONMASKABLE INTERRUPT (NMI)

The nonmaskable interrupt (NMI) can be generated in either of two ways:

- External hardware asserts the NMI pin.
- The processor receives a message on the system bus (Pentium 4, Intel Core Duo, Intel Core 2, Intel Atom, and Intel Xeon processors) or the APIC serial bus (P6 family and Pentium processors) with a delivery mode NMI.

When the processor receives a NMI from either of these sources, the processor handles it immediately by calling the NMI handler pointed to by interrupt vector number 2. The processor also invokes certain hardware conditions to ensure that no other interrupts, including NMI interrupts, are received until the NMI handler has completed executing (see Section 7.7.1, “Handling Multiple NMIs”).

Also, when an NMI is received from either of the above sources, it cannot be masked by the IF flag in the EFLAGS register.

It is possible to issue a maskable hardware interrupt (through the INTR pin) to vector 2 to invoke the NMI interrupt handler; however, this interrupt will not truly be an NMI interrupt. A true NMI interrupt that activates the processor’s NMI-handling hardware can only be delivered through one of the mechanisms listed above.

7.7.1 Handling Multiple NMIs

While an NMI interrupt handler is executing, the processor blocks delivery of subsequent NMIs until the next execution of the IRET instruction. This blocking of NMIs prevents nested execution of the NMI handler. It is recommended that the NMI interrupt handler be accessed through an interrupt gate to disable maskable hardware interrupts (see Section 7.8.1, “Masking Maskable Hardware Interrupts”).

An execution of the IRET instruction unblocks NMIs even if the instruction causes a fault. For example, if the IRET instruction executes with EFLAGS.VM = 1 and IOPL of less than 3, a general-protection exception is generated (see Section 23.2.7, “Sensitive Instructions”). In such a case, NMIs are unmasked before the exception handler is invoked.

When FRED transitions are enabled, FRED event delivery sets a bit in the stack frame when delivering an NMI. The subsequent return instruction (ERETS or ERETU) uses that bit to determine whether to unblock NMIs. See Chapter 8, “Flexible Return and Event Delivery (FRED).”

7.8 ENABLING AND DISABLING INTERRUPTS

The processor inhibits the generation of some interrupts, depending on the state of the processor and of the IF and RF flags in the EFLAGS register, as described in the following sections.

7.8.1 Masking Maskable Hardware Interrupts

The IF flag can disable the servicing of maskable hardware interrupts received on the processor's INTR pin or through the local APIC (see Section 7.3.2, "Maskable Hardware Interrupts"). When the IF flag is clear, the processor inhibits interrupts delivered to the INTR pin or through the local APIC from generating an internal interrupt request; when the IF flag is set, interrupts delivered to the INTR or through the local APIC pin are processed as normal external interrupts.

The IF flag does not affect non-maskable interrupts (NMIs) delivered to the NMI pin or delivery mode NMI messages delivered through the local APIC, nor does it affect processor generated exceptions. As with the other flags in the EFLAGS register, the processor clears the IF flag in response to a hardware reset.

The fact that the group of maskable hardware interrupts includes the reserved interrupt and exception vectors 0 through 32 can potentially cause confusion. Architecturally, when the IF flag is set, an interrupt for any of the vectors from 0 through 32 can be delivered to the processor through the INTR pin and any of the vectors from 16 through 32 can be delivered through the local APIC. The processor will then generate an interrupt and call the interrupt or exception handler pointed to by the vector number. So for example, it is possible to invoke the page-fault handler through the INTR pin (by means of vector 14); however, this is not a true page-fault exception. It is an interrupt. As with the INT *n* instruction (see Section 7.4.2, "Software-Generated Exceptions"), when an interrupt is generated through the INTR pin to an exception vector, the processor does not push an error code on the stack, so the exception handler may not operate correctly.

The IF flag can be set or cleared with the STI (set interrupt-enable flag) and CLI (clear interrupt-enable flag) instructions, respectively. These instructions may be executed only if the CPL is equal to or less than the IOPL. A general-protection exception (#GP) is generated if they are executed when the CPL is greater than the IOPL.¹ If IF = 0, maskable hardware interrupts remain inhibited on the instruction boundary following an execution of STI.² The inhibition ends after delivery of another event (e.g., exception) or the execution of the next instruction.

The IF flag is also affected by the following operations:

- The PUSHF instruction stores all flags on the stack, where they can be examined and modified. The POPF instruction can be used to load the modified flags back into the EFLAGS register.
- Task switches and the POPF, IRET, ERETS, and ERETU instructions load the EFLAGS register; therefore, they can be used to modify the setting of the IF flag.
- When an interrupt is handled through an interrupt gate, the IF flag is automatically cleared, which disables maskable hardware interrupts. (If an interrupt is handled through a trap gate, the IF flag is not cleared.) FRED event delivery of any event clears the IF flag.

See the descriptions of the CLI, STI, PUSHF, POPF, IRET, ERETS, and ERETU instructions in Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, and Chapter 4, "Instruction Set Reference, M-U," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B, for a detailed description of the operations these instructions are allowed to perform on the IF flag.

7.8.2 Masking Instruction Breakpoints

The RF (resume) flag in the EFLAGS register controls the response of the processor to instruction-breakpoint conditions (see the description of the RF flag in Section 2.3, "System Flags and Fields in the EFLAGS Register").

When set, it prevents an instruction breakpoint from generating a debug exception (#DB); when clear, instruction breakpoints will generate debug exceptions. The primary function of the RF flag is to prevent the processor from going into a debug exception loop on an instruction-breakpoint. See Section 20.3.1.1, "Instruction-Breakpoint Exception Condition," for more information on the use of this flag.

1. The effect of the IOPL on these instructions is modified slightly when the virtual mode extension is enabled by setting the VME flag in control register CR4: see Section 23.3, "Interrupt and Exception Handling in Virtual-8086 Mode." Behavior is also impacted by the PVI flag: see Section 23.4, "Protected-Mode Virtual Interrupts."

2. Nonmaskable interrupts and system-management interrupts may also be inhibited on the instruction boundary following such an execution of STI.

As noted in Section 7.8.3, execution of the MOV or POP instruction to load the SS register suppresses any instruction breakpoint on the next instruction (just as if EFLAGS.RF were 1).

7.8.3 Masking Exceptions and Interrupts When Switching Stacks

To switch to a different stack segment, software often uses a pair of instructions, for example:

```
MOV SS, AX
MOV ESP, StackTop
```

(Software might also use the POP instruction to load SS and ESP.)

If an interrupt or exception occurs after the new SS segment descriptor has been loaded but before the ESP register has been loaded, these two parts of the logical address into the stack space are inconsistent for the duration of the interrupt or exception handler (assuming that delivery of the interrupt or exception does not itself load a new stack pointer).

To account for this situation, the processor prevents certain events from being delivered after execution of a MOV to SS instruction or a POP to SS instruction. The following items provide details:

- Any instruction breakpoint on the next instruction is suppressed (as if EFLAGS.RF were 1).
- Any data breakpoint on the MOV to SS instruction or POP to SS instruction is inhibited until the instruction boundary following the next instruction.
- Any single-step trap that would be delivered following the MOV to SS instruction or POP to SS instruction (because EFLAGS.TF is 1) is suppressed.
- The suppression and inhibition ends after delivery of an exception or the execution of the next instruction.
- If a sequence of consecutive instructions each loads the SS register (using MOV or POP), only the first is guaranteed to inhibit or suppress events in this way.

Intel recommends that software use the LSS instruction to load the SS register and ESP together. The problem identified earlier does not apply to LSS, and the LSS instruction does not inhibit events as detailed above.

7.9 PRIORITIZATION OF CONCURRENT EVENTS

If more than one event is pending at an instruction boundary (between execution of instructions), the processor services them in a predictable order. Table 7-2 shows the priority among classes of event sources.

Table 7-2. Priority Among Concurrent Events

Priority	Description
1 (Highest)	Hardware Reset and Machine Checks - RESET - Machine Check (#MC)
2	Trap on Task Switch - T flag in TSS is set (#DB)
3	External Hardware Interventions - FLUSH - STOPCLK - SMI - INIT
4	Traps on the Previous Instruction - Trap-class Debug Exceptions (#DB due to TF flag set or data/I-O breakpoint)
5	Nonmaskable Interrupts (NMI) ¹
6	Maskable Hardware Interrupts ¹

Table 7-2. Priority Among Concurrent Events (Contd.)

Priority	Description
7	Fault-class Debug Exceptions (#DB due to instruction breakpoint)
8	Faults from Fetching Next Instruction - Code-Segment Limit Violation (#GP) - Code Page Fault (#PF)
9 (Lowest)	Faults from Decoding the Next Instruction - Control protection exception due to missing ENDBRANCH at target of an indirect call or jump (#CP) - Instruction length > 15 bytes (#GP) - Invalid Opcode (#UD) - Coprocessor Not Available (#NM)

NOTE

1. The Intel® 486 processor and earlier processors group nonmaskable and maskable interrupts in the same priority class.

The processor first services a pending event from the class which has the highest priority, transferring execution to the first instruction of the handler. Lower priority exceptions are discarded; lower priority interrupts are held pending. Discarded exceptions may be re-generated when the event handler returns execution to the point in the program or task where the original event occurred. While the priority among the classes listed in Table 7-2 is consistent across processor implementations, the priority of events within a class is implementation-dependent and may vary from processor to processor.

Table 7-2 specifies the prioritization of events that may be pending at an instruction boundary. It does not specify the prioritization of faults that arise during instruction execution or event delivery (these include #BR, #TS, #NP, #SS, #GP, #PF, #AC, #MF, #XM, #VE, or #CP). It also does not apply to the events generated by the “Call to Interrupt Procedure” instructions (INT n, INTO, INT3, and INT1), as these events are integral to the execution of those instructions and do not occur between instructions.

7.10 INTERRUPT DESCRIPTOR TABLE (IDT)

NOTE

FRED event delivery is an alternative to use of the IDT. The material in this section does not apply when FRED transitions are enabled. See Chapter 8, “Flexible Return and Event Delivery (FRED),” for more information about FRED transitions.

The interrupt descriptor table (IDT) associates each exception or interrupt vector with a gate descriptor for the procedure or task used to service the associated exception or interrupt. Like the GDT and LDTs, the IDT is an array of 8-byte descriptors (in protected mode). Unlike the GDT, the first entry of the IDT may contain a descriptor. To form an index into the IDT, the processor scales the exception or interrupt vector by eight (the number of bytes in a gate descriptor). Because there are only 256 interrupt or exception vectors, the IDT need not contain more than 256 descriptors. It can contain fewer than 256 descriptors, because descriptors are required only for the interrupt and exception vectors that may occur. All empty descriptor slots in the IDT should have the present flag for the descriptor set to 0.

The base addresses of the IDT should be aligned on an 8-byte boundary to maximize performance of cache line fills. The limit value is expressed in bytes and is added to the base address to get the address of the last valid byte. A limit value of 0 results in exactly 1 valid byte. Because IDT entries are always eight bytes long, the limit should always be one less than an integral multiple of eight (that is, $8N - 1$).

The IDT may reside anywhere in the linear address space. As shown in Figure 7-1, the processor locates the IDT using the IDTR register. This register holds both a 32-bit base address and 16-bit limit for the IDT.

The LIDT (load IDT register) and SIDT (store IDT register) instructions load and store the contents of the IDTR register, respectively. The LIDT instruction loads the IDTR register with the base address and limit held in a memory operand. This instruction can be executed only when the CPL is 0. It normally is used by the initialization code of an operating system when creating an IDT. An operating system also may use it to change from one IDT to

another. The SIDT instruction copies the base and limit value stored in IDTR to memory. This instruction can be executed at any privilege level.

If a vector references a descriptor beyond the limit of the IDT, a general-protection exception (#GP) is generated.

NOTE

Because interrupts are delivered to the processor core only once, an incorrectly configured IDT could result in incomplete interrupt handling and/or the blocking of interrupt delivery.

IA-32 architecture rules need to be followed for setting up IDTR base/limit/access fields and each field in the gate descriptors. The same apply for the Intel 64 architecture. This includes implicit referencing of the destination code segment through the GDT or LDT and accessing the stack.

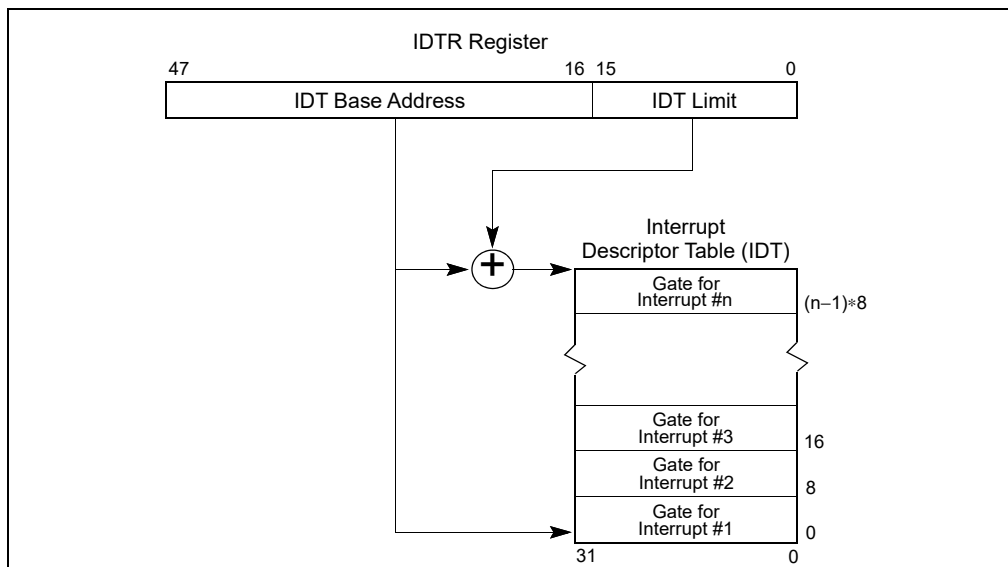


Figure 7-1. Relationship of the IDTR and IDT

7.11 IDT DESCRIPTORS

NOTE

FRED event delivery does not use IDT descriptors. The material in this section does not apply when FRED transitions are enabled. See Chapter 8, "Flexible Return and Event Delivery (FRED)," for more information about FRED transitions.

The IDT may contain any of three kinds of gate descriptors:

- Task-gate descriptor
- Interrupt-gate descriptor
- Trap-gate descriptor

Figure 7-2 shows the formats for the task-gate, interrupt-gate, and trap-gate descriptors. The format of a task gate used in an IDT is the same as that of a task gate used in the GDT or an LDT (see Section 10.2.5, "Task-Gate Descriptor"). The task gate contains the segment selector for a TSS for an exception and/or interrupt handler task.

Interrupt and trap gates are very similar to call gates (see Section 6.8.3, "Call Gates"). They contain a far pointer (segment selector and offset) that the processor uses to transfer program execution to a handler procedure in an exception- or interrupt-handler code segment. These gates differ in the way the processor handles the IF flag in the EFLAGS register (see Section 7.12.1.3, "Flag Usage By Exception- or Interrupt-Handler Procedure").

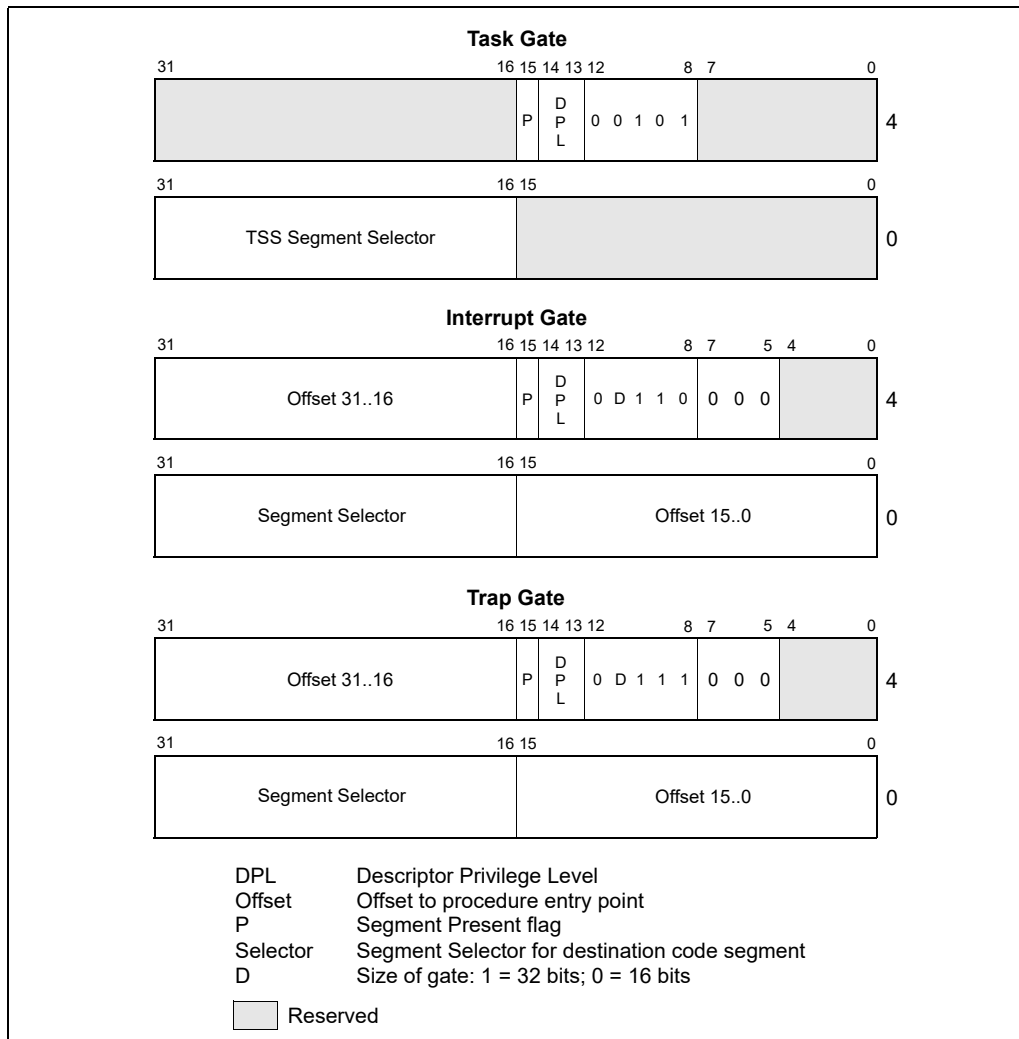


Figure 7-2. IDT Gate Descriptors

7.12 EXCEPTION AND INTERRUPT HANDLING

NOTE

This section focuses on event handling based on IDT event delivery and on return using the IRET instruction. FRED transitions are alternatives to these mechanisms. In general, the material in this section does not apply when FRED transitions are enabled. See Chapter 8, “Flexible Return and Event Delivery (FRED),” for more information about FRED transitions.

The processor handles calls to exception- and interrupt-handlers similar to the way it handles calls with a CALL instruction to a procedure or a task. When responding to an exception or interrupt, the processor uses the exception or interrupt vector as an index to a descriptor in the IDT. If the index points to an interrupt gate or trap gate, the processor calls the exception or interrupt handler in a manner similar to a CALL to a call gate (see Section 6.8.2, “Gate Descriptors,” through Section 6.8.6, “Returning from a Called Procedure”). If index points to a task gate, the processor executes a task switch to the exception- or interrupt-handler task in a manner similar to a CALL to a task gate (see Section 10.3, “Task Switching”).

7.12.1 Exception- or Interrupt-Handler Procedures

An interrupt gate or trap gate references an exception- or interrupt-handler procedure that runs in the context of the currently executing task (see Figure 7-3). The segment selector for the gate points to a segment descriptor for an executable code segment in either the GDT or the current LDT. The offset field of the gate descriptor points to the beginning of the exception- or interrupt-handling procedure.

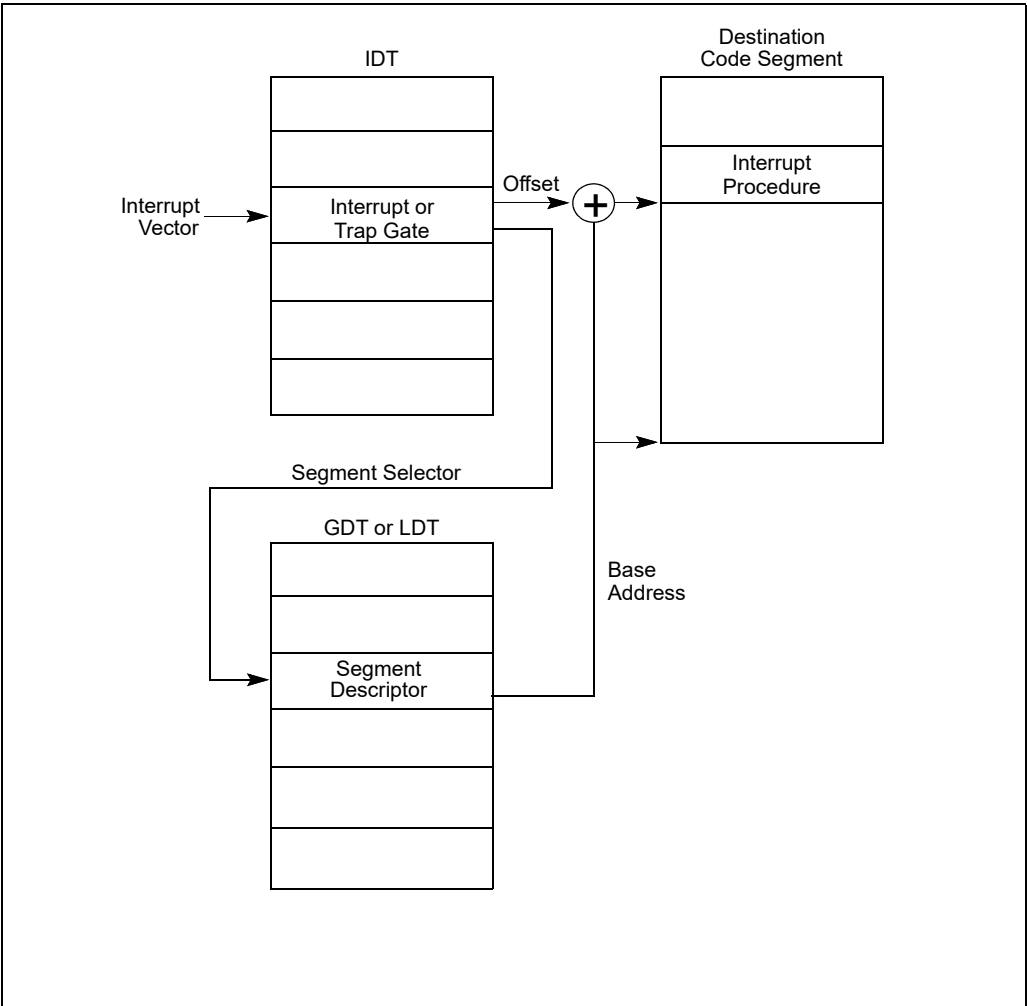


Figure 7-3. Interrupt Procedure Call

When the processor performs a call to the exception- or interrupt-handler procedure:

- If the handler procedure is going to be executed at a numerically lower privilege level, a stack switch occurs. When the stack switch occurs:
 - a. The segment selector and stack pointer for the stack to be used by the handler are obtained from the TSS for the currently executing task. On this new stack, the processor pushes the stack segment selector and stack pointer of the interrupted procedure.
 - b. The processor then saves the current state of the EFLAGS, CS, and EIP registers on the new stack (see Figure 7-4).
 - c. If an exception causes an error code to be saved, it is pushed on the new stack after the EIP value.
- If the handler procedure is going to be executed at the same privilege level as the interrupted procedure:
 - a. The processor saves the current state of the EFLAGS, CS, and EIP registers on the current stack (see Figure 7-4).
 - b. If an exception causes an error code to be saved, it is pushed on the current stack after the EIP value.

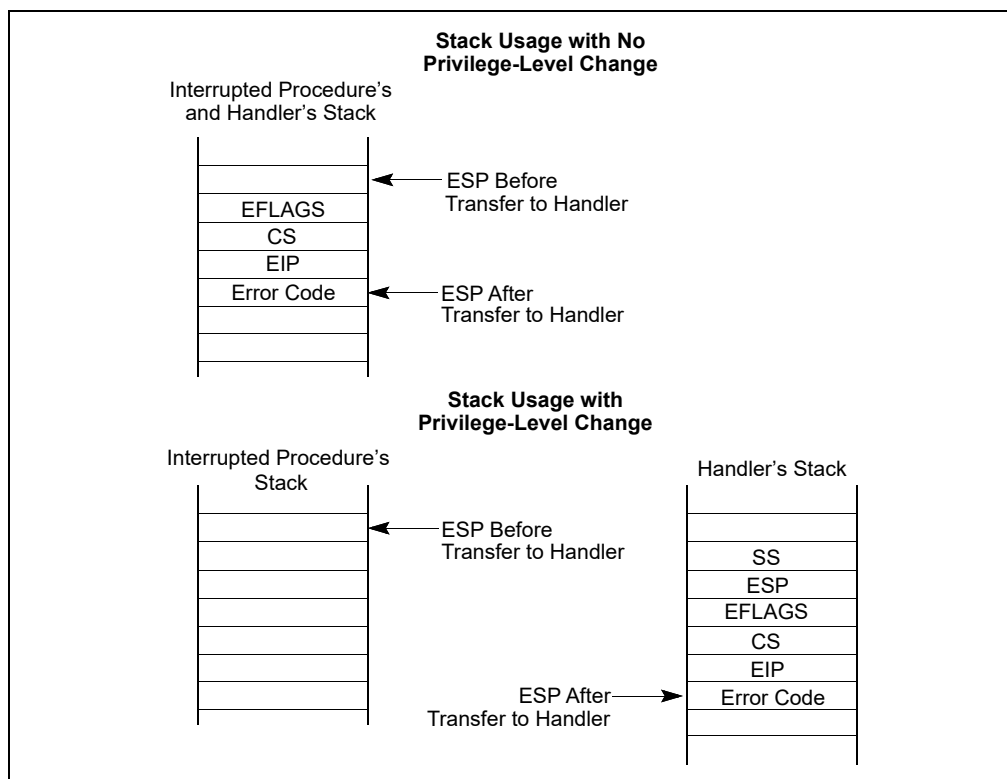


Figure 7-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

To return from an exception- or interrupt-handler procedure, the handler must use the IRET (or IRETD) instruction. The IRET instruction is similar to the RET instruction except that it restores the saved flags into the EFLAGS register. The IOPL field of the EFLAGS register is restored only if the CPL is 0. The IF flag is changed only if the CPL is less than or equal to the IOPL. See Chapter 3, "Instruction Set Reference, A-L," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for a description of the complete operation performed by the IRET instruction.

If a stack switch occurred when calling the handler procedure, the IRET instruction switches back to the interrupted procedure's stack on the return.

7.12.1.1 Shadow Stack Usage on Transfers to Interrupt and Exception Handling Routines

When the processor performs a call to the exception- or interrupt-handler procedure:

- If the handler procedure is going to be executed at a numerically lower privilege level, a shadow stack switch occurs. When the shadow stack switch occurs:
 - a. On a transfer from privilege level 3, if shadow stacks are enabled at privilege level 3 then the SSP is saved to the IA32_PL3_SSP MSR.
 - b. If shadow stacks are enabled at the privilege level where the handler will execute then the shadow stack for the handler is obtained from one of the following MSRs based on the privilege level at which the handler executes.
 - IA32_PL2_SSP if handler executes at privilege level 2.
 - IA32_PL1_SSP if handler executes at privilege level 1.
 - IA32_PL0_SSP if handler executes at privilege level 0.
 - c. The SSP obtained is then verified to ensure it points to a valid supervisory shadow stack that is not currently active by verifying a supervisor shadow stack token at the address pointed to by the SSP. The operations performed to verify and acquire the supervisor shadow stack token by making it busy are as described in Section 18.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.
 - d. On this new shadow stack, the processor pushes the CS, LIP (CS.base + EIP), and SSP of the interrupted procedure if the interrupted procedure was executing at privilege level less than 3; see Figure 7-5.¹
- If the handler procedure is going to be executed at the same privilege level as the interrupted procedure and shadow stacks are enabled at current privilege level:
 - a. The processor saves the current state of the CS, LIP (CS.base + EIP), and SSP registers on the current shadow stack; see Figure 7-5.

1. If any of these pushes leads to an exception or a VM exit, the supervisor shadow-stack token remains busy.

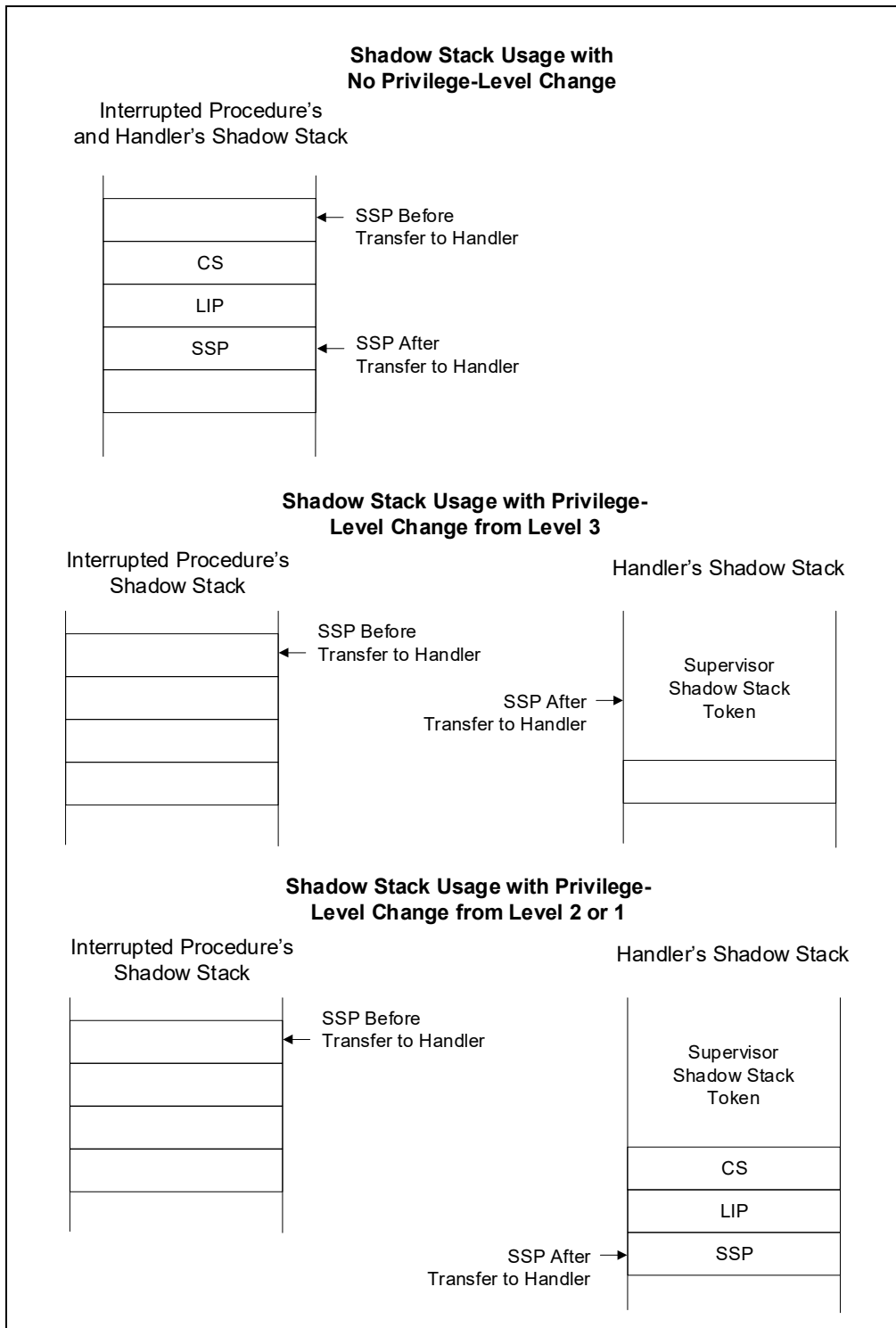


Figure 7-5. Shadow Stack Usage on Transfers to Interrupt and Exception-Handling Routines

To return from an exception- or interrupt-handler procedure, the handler must use the IRET (or IRETD) instruction. When executing a return from an interrupt or exception handler from the same privilege level as the interrupted procedure, the processor performs these actions to enforce return address protection:

- Restores the CS and EIP registers to their values prior to the interrupt or exception.

If shadow stack is enabled:

- Compares the values on shadow stack at address $SSP+8$ (the LIP) and $SSP+16$ (the CS) to the CS and $(CS.base + EIP)$ popped from the stack and causes a control protection exception ($\#CP(FAR-RET/IRET)$) if they do not match.
- Pops the top-of-stack value (the SSP prior to the interrupt or exception) from shadow stack into SSP register.

When executing a return from an interrupt or exception handler from a different privilege level than the interrupted procedure, the processor performs the actions below.

- If shadow stack is enabled at current privilege level:
 - If SSP is not aligned to 8 bytes then causes a control protection exception ($\#CP(FAR-RET/IRET)$).
 - If privilege level of the procedure being returned to is less than 3 (returning to supervisor mode):
 - Compares the values on shadow stack at address $SSP+8$ (the LIP) and $SSP+16$ (the CS) to the CS and $(CS.base + EIP)$ popped from the stack and causes a control protection exception ($\#CP(FAR-RET/IRET)$) if they do not match.
 - Temporarily saves the top-of-stack value (the SSP of the procedure being returned to) internally.
 - If a busy supervisor shadow stack token is present at address $SSP+24$, then marks the token free using operations described in section Section 18.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.
 - If the privilege level of the procedure being returned to is less than 3 (returning to supervisor mode), restores the SSP register from the internally saved value.
 - If the privilege level of the procedure being returned to is 3 (returning to user mode) and shadow stack is enabled at privilege level 3, then restores the SSP register with value of $IA32\_PL3\_SSP$ MSR.

7.12.1.2 Protection of Exception- and Interrupt-Handler Procedures

The privilege-level protection for exception- and interrupt-handler procedures is similar to that used for ordinary procedure calls when called through a call gate (see Section 6.8.4, "Accessing a Code Segment Through a Call Gate"). The processor does not permit transfer of execution to an exception- or interrupt-handler procedure in a less privileged code segment (numerically greater privilege level) than the CPL.

An attempt to violate this rule results in a general-protection exception ($\#GP$). The protection mechanism for exception- and interrupt-handler procedures is different in the following ways:

- Because interrupt and exception vectors have no RPL, the RPL is not checked on implicit calls to exception and interrupt handlers.
- The processor checks the DPL of the interrupt or trap gate only if an exception or interrupt is generated with an $INT\ n$, $INT3$, or $INTO$ instruction.¹ Here, the CPL must be less than or equal to the DPL of the gate. This restriction prevents application programs or procedures running at privilege level 3 from using a software interrupt to access critical exception handlers, such as the page-fault handler, providing that those handlers are placed in more privileged code segments (numerically lower privilege level). For hardware-generated interrupts and processor-detected exceptions, the processor ignores the DPL of interrupt and trap gates.

Because exceptions and interrupts generally do not occur at predictable times, these privilege rules effectively impose restrictions on the privilege levels at which exception and interrupt- handling procedures can run. Either of the following techniques can be used to avoid privilege-level violations.

- The exception or interrupt handler can be placed in a conforming code segment. This technique can be used for handlers that only need to access data available on the stack (for example, divide error exceptions). If the handler needs data from a data segment, the data segment needs to be accessible from privilege level 3, which would make it unprotected.
- The handler can be placed in a nonconforming code segment with privilege level 0. This handler would always run, regardless of the CPL that the interrupted program or task is running at.

1. This check is not performed by execution of the $INT1$ instruction (opcode $F1$); it would be performed by execution of $INT\ 1$ (opcode $CD\ 01$).

7.12.1.3 Flag Usage By Exception- or Interrupt-Handler Procedure

When accessing an exception or interrupt handler through either an interrupt gate or a trap gate, the processor clears the TF flag in the EFLAGS register after it saves the contents of the EFLAGS register on the stack. (On calls to exception and interrupt handlers, the processor also clears the VM, RF, and NT flags in the EFLAGS register, after they are saved on the stack.) Clearing the TF flag prevents instruction tracing from affecting interrupt response and ensures that no single-step exception will be delivered after delivery to the handler. A subsequent IRET instruction restores the TF (and VM, RF, and NT) flags to the values in the saved contents of the EFLAGS register on the stack.

The only difference between an interrupt gate and a trap gate is the way the processor handles the IF flag in the EFLAGS register. When accessing an exception- or interrupt-handling procedure through an interrupt gate, the processor clears the IF flag to prevent other interrupts from interfering with the current interrupt handler. A subsequent IRET instruction restores the IF flag to its value in the saved contents of the EFLAGS register on the stack. Accessing a handler procedure through a trap gate does not affect the IF flag.

7.12.2 Interrupt Tasks

When an exception or interrupt handler is accessed through a task gate in the IDT, a task switch results. Handling an exception or interrupt with a separate task offers several advantages:

- The entire context of the interrupted program or task is saved automatically.
- A new TSS permits the handler to use a new privilege level 0 stack when handling the exception or interrupt. If an exception or interrupt occurs when the current privilege level 0 stack is corrupted, accessing the handler through a task gate can prevent a system crash by providing the handler with a new privilege level 0 stack.
- The handler can be further isolated from other tasks by giving it a separate address space. This is done by giving it a separate LDT.

The disadvantage of handling an interrupt with a separate task is that the amount of machine state that must be saved on a task switch makes it slower than using an interrupt gate, resulting in increased interrupt latency.

A task gate in the IDT references a TSS descriptor in the GDT (see Figure 7-6). A switch to the handler task is handled in the same manner as an ordinary task switch (see Section 10.3, “Task Switching”). The link back to the interrupted task is stored in the previous task link field of the handler task’s TSS. If an exception caused an error code to be generated, this error code is copied to the stack of the new task.

When exception- or interrupt-handler tasks are used in an operating system, there are actually two mechanisms that can be used to dispatch tasks: the software scheduler (part of the operating system) and the hardware scheduler (part of the processor's interrupt mechanism). The software scheduler needs to accommodate interrupt tasks that may be dispatched when interrupts are enabled.

NOTE

Because IA-32 architecture tasks are not re-entrant, an interrupt-handler task must disable interrupts between the time it completes handling the interrupt and the time it executes the IRET instruction. This action prevents another interrupt from occurring while the interrupt task's TSS is still marked busy, which would cause a general-protection (#GP) exception.

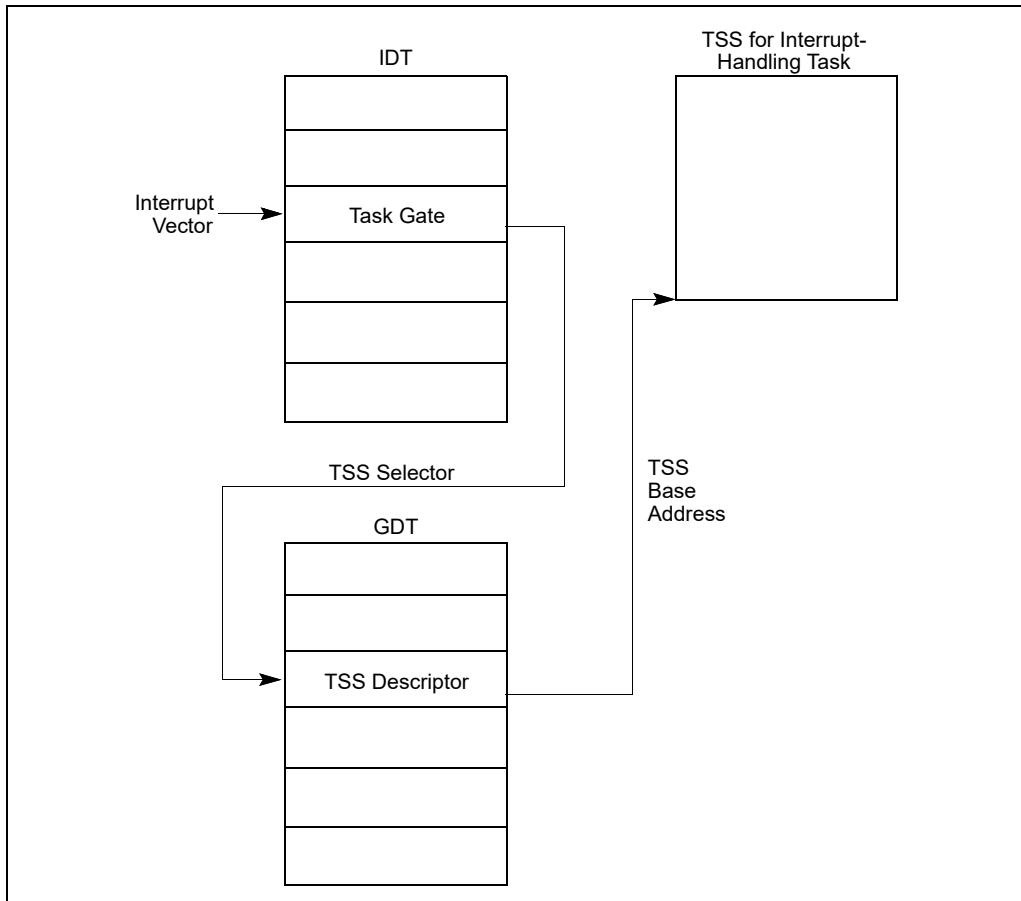


Figure 7-6. Interrupt Task Switch

7.13 ERROR CODE

When an exception condition is related to a specific segment selector or IDT vector, the processor pushes an error code onto the stack of the exception handler (whether it is a procedure or task). The error code has the format shown in Figure 7-7. The error code resembles a segment selector; however, instead of a TI flag and RPL field, the error code contains 3 flags:

- EXT** **External event (bit 0)** — When set, indicates that the exception occurred during delivery of an event external to the program, such as an interrupt or an earlier exception.¹ The bit is cleared if the exception occurred during delivery of a software interrupt (INT *n*, INT3, or INTO).
- IDT** **Descriptor location (bit 1)** — When set, indicates that the index portion of the error code refers to a gate descriptor in the IDT; when clear, indicates that the index refers to a descriptor in the GDT or the current LDT. *The processor does not set this bit when FRED transitions are enabled.*
- TI** **GDT/LDT (bit 2)** — Only used when the IDT flag is clear. When set, the TI flag indicates that the index portion of the error code refers to a segment or gate descriptor in the LDT; when clear, it indicates that the index refers to a descriptor in the current GDT.

1. The bit is also set if the exception occurred during delivery of INT1.

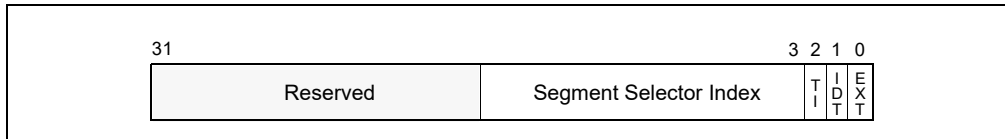


Figure 7-7. Error Code

The segment selector index field provides an index into the IDT, GDT, or current LDT to the segment or gate selector being referenced by the error code. In some cases the error code is null (all bits are clear except possibly EXT). A null error code indicates that the error was not caused by a reference to a specific segment or that a null segment selector was referenced in an operation.

The format of the error code is different for page-fault exceptions (#PF). See the “Interrupt 14—Page-Fault Exception (#PF)” section in this chapter.

The format of the error code is different for control protection exceptions (#CP). See the “Interrupt 21—Control Protection Exception (#CP)” section in this chapter.

The error code is pushed on the stack as a doubleword or word (depending on the default interrupt, trap, or task gate size). To keep the stack aligned for doubleword pushes, the upper half of the error code is reserved. Note that the error code is not popped when the IRET instruction is executed to return from an exception handler, so the handler must remove the error code before executing a return.

Error codes are not pushed on the stack for exceptions that are generated externally (with the INTR or LINT[1:0] pins) or the INT *n* instruction, even if an error code is normally produced for those exceptions.

7.14 EXCEPTION AND INTERRUPT HANDLING IN 64-BIT MODE

NOTE

This section focuses on event handling based on IDT event delivery and on return using the IRET instruction. FRED transitions are alternatives to these mechanisms. In general, the material in this section does not apply when FRED transitions are enabled. See Chapter 8, “Flexible Return and Event Delivery (FRED),” for more information about FRED transitions.

In 64-bit mode, interrupt and exception handling is similar to what has been described for non-64-bit modes. The following are the exceptions:

- All interrupt handlers pointed by the IDT are in 64-bit code (this does not apply to the SMI handler).
- The size of interrupt-stack pushes is fixed at 64 bits; and the processor uses 8-byte, zero extended stores.
- The stack pointer (SS:RSP) is pushed unconditionally on interrupts. In legacy modes, this push is conditional and based on a change in current privilege level (CPL).
- The new SS is set to NULL if there is a change in CPL.
- IRET behavior changes.
- There is a new interrupt stack-switch mechanism and a new interrupt shadow stack-switch mechanism.
- The alignment of interrupt stack frame is different.

7.14.1 64-Bit Mode IDT

Interrupt and trap gates are 16 bytes in length to provide a 64-bit offset for the instruction pointer (RIP). The 64-bit RIP referenced by interrupt-gate descriptors allows an interrupt service routine to be located anywhere in the linear-address space. See Figure 7-8.

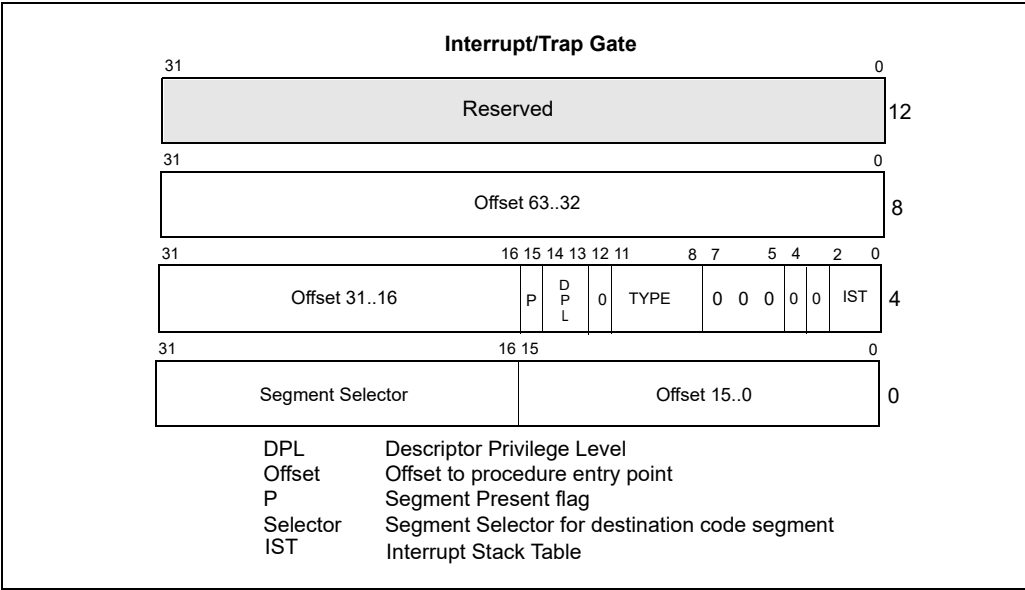


Figure 7-8. 64-Bit IDT Gate Descriptors

In 64-bit mode, the IDT index is formed by scaling the interrupt vector by 16. The first eight bytes (bytes 7:0) of a 64-bit mode interrupt gate are similar but not identical to legacy 32-bit interrupt gates. The type field (bits 11:8 in bytes 7:4) is described in Table 3-2. The Interrupt Stack Table (IST) field (bits 4:0 in bytes 7:4) is used by the stack switching mechanisms described in Section 7.14.5, “Interrupt Stack Table.” Bytes 11:8 hold the upper 32 bits of the target RIP (interrupt segment offset) in canonical form. A general-protection exception (#GP) is generated if software attempts to reference an interrupt gate with a target RIP that is not in canonical form.

The target code segment referenced by the interrupt gate must be a 64-bit code segment (CS.L = 1, CS.D = 0). If the target is not a 64-bit code segment, a general-protection exception (#GP) is generated with the IDT vector number reported as the error code.

Only 64-bit interrupt and trap gates can be referenced in IA-32e mode (64-bit mode and compatibility mode). Legacy 32-bit interrupt or trap gate types (0EH or 0FH) are redefined in IA-32e mode as 64-bit interrupt and trap gate types. No 32-bit interrupt or trap gate type exists in IA-32e mode. If a reference is made to a 16-bit interrupt or trap gate (06H or 07H), a general-protection exception (#GP(0)) is generated.

7.14.2 64-Bit Mode Stack Frame

In legacy mode, the size of an IDT entry (16 bits or 32 bits) determines the size of interrupt-stack-frame pushes. SS:ESP is pushed only on a CPL change. In 64-bit mode, the size of interrupt stack-frame pushes is fixed at eight bytes. This is because only 64-bit mode gates can be referenced. 64-bit mode also pushes SS:RSP unconditionally, rather than only on a CPL change.

When shadow stacks are enabled at the interrupt handler’s privilege level and the interrupted procedure was not executing at a privilege level 3, then the processor pushes the CS:LIP:SSP of the interrupted procedure on the shadow stack of the interrupt handler (where LIP is the linear address of the return address).

Aside from error codes, pushing SS:RSP unconditionally presents operating systems with a consistent interrupt-stackframe size across all interrupts. Interrupt service-routine entry points that handle interrupts generated by the INTn instruction or external INTR# signal can push an additional error code place-holder to maintain consistency.

In legacy mode, the stack pointer may be at any alignment when an interrupt or exception causes a stack frame to be pushed. This causes the stack frame and succeeding pushes done by an interrupt handler to be at arbitrary alignments. In IA-32e mode, the RSP is aligned to a 16-byte boundary before pushing the stack frame. The stack frame itself is aligned on a 16-byte boundary when the interrupt handler is called. The processor can arbitrarily realign the new RSP on interrupts because the previous (possibly unaligned) RSP is unconditionally saved on the newly aligned stack. The previous RSP will be automatically restored by a subsequent IRET.

Aligning the stack permits exception and interrupt frames to be aligned on a 16-byte boundary before interrupts are re-enabled. This allows the stack to be formatted for optimal storage of 16-byte XMM registers, which enables the interrupt handler to use faster 16-byte aligned loads and stores (MOVAPS rather than MOVUPS) to save and restore XMM registers.

Although the RSP alignment is always performed when LMA = 1, it is only of consequence for the kernel-mode case where there is no stack switch or IST used. For a stack switch or IST, the OS would have presumably put suitably aligned RSP values in the TSS.

7.14.3 IRET in IA-32e Mode

In IA-32e mode, IRET executes with an 8-byte operand size. There is nothing that forces this requirement. The stack is formatted in such a way that for actions where IRET is required, the 8-byte IRET operand size works correctly.

Because interrupt stack-frame pushes are always eight bytes in IA-32e mode, an IRET must pop eight byte items off the stack. This is accomplished by preceding the IRET with a 64-bit operand-size prefix. The size of the pop is determined by the address size of the instruction. The SS/ESP/RSP size adjustment is determined by the stack size.

IRET pops SS:RSP unconditionally off the interrupt stack frame only when it is executed in 64-bit mode. In compatibility mode, IRET pops SS:RSP off the stack only if there is a CPL change. This allows legacy applications to execute properly in compatibility mode when using the IRET instruction. 64-bit interrupt service routines that exit with an IRET unconditionally pop SS:RSP off of the interrupt stack frame, even if the target code segment is running in 64-bit mode or at CPL = 0. This is because the original interrupt always pushes SS:RSP.

When shadow stacks are enabled and the target privilege level is not 3, the CS:LIP from the shadow stack frame is compared to the return linear address formed by CS:EIP from the stack. If they do not match then the processor caused a control protection exception (#CP(FAR-RET/IRET)), else the processor pops the SSP of the interrupted procedure from the shadow stack. If the target privilege level is 3 and shadow stacks are enabled at privilege level 3, then the SSP for the interrupted procedure is restored from the IA32_PL3_SSP MSR.

In IA-32e mode, IRET is allowed to load a NULL SS under certain conditions. If the target mode is 64-bit mode and the target CPL $\geq$ 3, IRET allows SS to be loaded with a NULL selector. As part of the stack switch mechanism, an interrupt or exception sets the new SS to NULL, instead of fetching a new SS selector from the TSS and loading the corresponding descriptor from the GDT or LDT. The new SS selector is set to NULL in order to properly handle returns from subsequent nested far transfers. If the called procedure itself is interrupted, the NULL SS is pushed on the stack frame. On the subsequent IRET, the NULL SS on the stack acts as a flag to tell the processor not to load a new SS descriptor.

7.14.4 Stack Switching in IA-32e Mode

The IA-32 architecture provides a mechanism to automatically switch stack frames in response to an interrupt. The 64-bit extensions of Intel 64 architecture implement a modified version of the legacy stack-switching mechanism and an alternative stack-switching mechanism called the interrupt stack table (IST).

In IA-32 modes, the legacy IA-32 stack-switch mechanism is unchanged. In IA-32e mode, the legacy stack-switch mechanism is modified. When stacks are switched as part of a 64-bit mode privilege-level change (resulting from an interrupt), a new SS descriptor is not loaded. IA-32e mode loads only an inner-level RSP from the TSS. The new SS selector is forced to NULL and the SS selector's RPL field is set to the new CPL. The new SS is set to NULL in order to handle nested far transfers (far CALL, INT, interrupts, and exceptions). The old SS and RSP are saved on the new stack (Figure 7-9). On the subsequent IRET, the old SS is popped from the stack and loaded into the SS register.

In summary, a stack switch in IA-32e mode works like the legacy stack switch, except that a new SS selector is not loaded from the TSS. Instead, the new SS is forced to NULL.

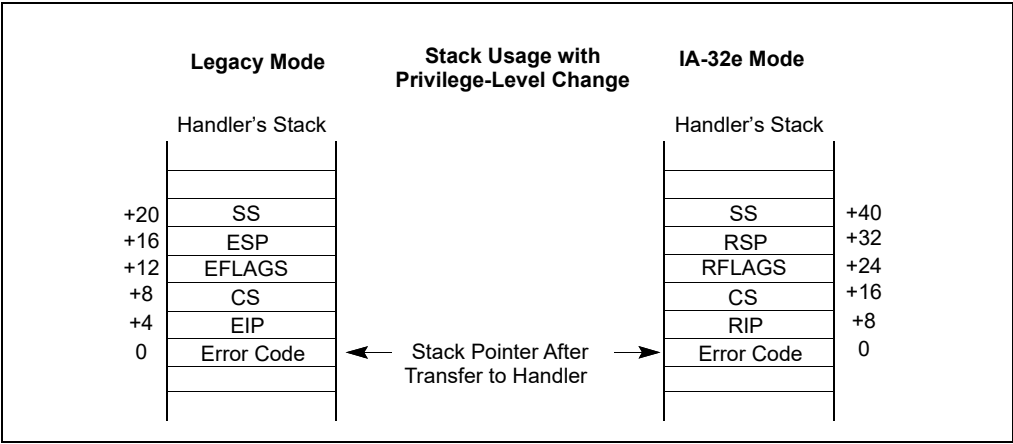


Figure 7-9. IA-32e Mode Stack Usage After Privilege Level Change

7.14.5 Interrupt Stack Table

In IA-32e mode, a new interrupt stack table (IST) mechanism is available as an alternative to the modified legacy stack-switching mechanism described above. This mechanism unconditionally switches stacks when it is enabled. It can be enabled on an individual interrupt-vector basis using a field in the IDT entry. This means that some interrupt vectors can use the modified legacy mechanism and others can use the IST mechanism.

The IST mechanism is only available in IA-32e mode. It is part of the 64-bit mode TSS. The motivation for the IST mechanism is to provide a method for specific interrupts (such as NMI, double-fault, and machine-check) to always execute on a known good stack. In legacy mode, interrupts can use the task-switch mechanism to set up a known-good stack by accessing the interrupt service routine through a task gate located in the IDT. However, the legacy task-switch mechanism is not supported in IA-32e mode.

The IST mechanism provides up to seven IST pointers in the TSS. The pointers are referenced by an interrupt-gate descriptor in the interrupt-descriptor table (IDT); see Figure 7-8. The gate descriptor contains a 3-bit IST index field that provides an offset into the IST section of the TSS. Using the IST mechanism, the processor loads the value pointed by an IST pointer into the RSP.

When an interrupt occurs, the new SS selector is forced to NULL and the SS selector’s RPL field is set to the new CPL. The old SS, RSP, RFLAGS, CS, and RIP are pushed onto the new stack. Interrupt processing then proceeds as normal. If the IST index is zero, the modified legacy stack-switching mechanism described above is used.

To support this stack-switching mechanism with shadow stacks enabled, the processor provides an MSR, `IA32_INTERRUPT_SSP_TABLE_ADDR`, to program the linear address of a table of seven shadow stack pointers that are selected using the IST index from the gate descriptor. To switch to a shadow stack selected from the interrupt shadow stack table pointed to by the `IA32_INTERRUPT_SSP_TABLE_ADDR`, the processor requires that the shadow stack addresses programmed into this table point to a supervisor shadow stack token; see Figure 7-10.

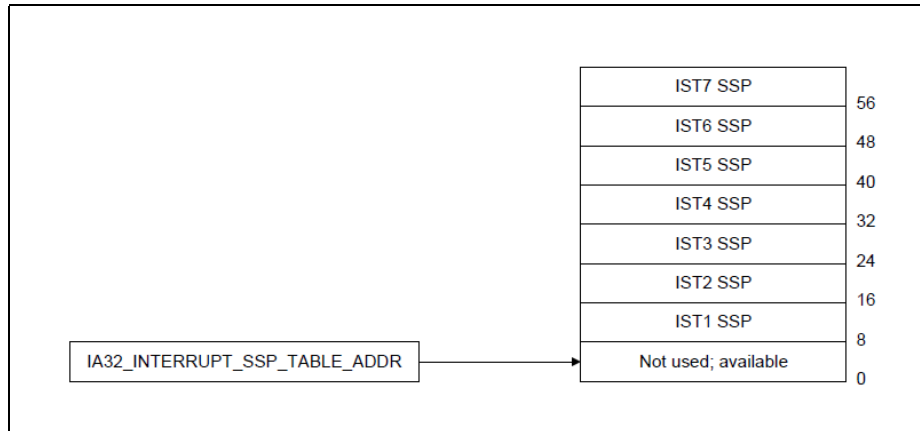


Figure 7-10. Interrupt Shadow Stack Table

7.15 EXCEPTION AND INTERRUPT REFERENCE

The following sections describe conditions which generate exceptions and interrupts. They are arranged in the order of vector numbers. The information contained in these sections are as follows:

- **Exception Class** — Indicates whether the exception class is a fault, trap, or abort type. Some exceptions can be either a fault or trap type, depending on when the error condition is detected. (This section is not applicable to interrupts.)
- **Description** — Gives a general description of the purpose of the exception or interrupt type. It also describes how the processor handles the exception or interrupt.
- **Exception Error Code** — Indicates whether an error code is saved for the exception. If one is saved, the contents of the error code are described. (This section is not applicable to interrupts.)
- **Saved Instruction Pointer** — Describes which instruction the saved (or return) instruction pointer points to. It also indicates whether the pointer can be used to restart a faulting instruction.
- **Program State Change** — Describes the effects of the exception or interrupt on the state of the currently running program or task and the possibilities of restarting the program or task without loss of continuity.

Interrupt 0—Divide Error Exception (#DE)

Exception Class **Fault.**

Description

Indicates the divisor operand for a DIV or IDIV instruction is 0 or that the result cannot be represented in the number of bits specified for the destination operand.

Exception Error Code

None.

Saved Instruction Pointer

Saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

A program-state change does not accompany the divide error, because the exception occurs before the faulting instruction is executed.

Interrupt 1—Debug Exception (#DB)

Exception Class **Trap or Fault.** The exception handler can distinguish between traps or faults by examining the contents of DR6 and the other debug registers.

Description

Indicates that one or more of several debug-exception conditions has been detected. Whether the exception is a fault or a trap depends on the condition (see Table 7-3). See Chapter 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features,” for detailed information about the debug exceptions.

Table 7-3. Debug Exception Conditions and Corresponding Exception Classes

Exception Condition	Exception Class
Instruction fetch breakpoint	Fault
Data read or write breakpoint	Trap
I/O read or write breakpoint	Trap
General detect condition (in conjunction with in-circuit emulation)	Fault
Single-step	Trap
Task-switch	Trap
Execution of INT1 ¹	Trap

NOTES:

1. Hardware vendors may use the INT1 instruction for hardware debug. For that reason, Intel recommends software vendors instead use the INT3 instruction for software breakpoints.

Exception Error Code

None. An exception handler can examine the debug registers to determine which condition caused the exception.

Saved Instruction Pointer

Fault — Saved contents of CS and EIP registers point to the instruction that generated the exception.

Trap — Saved contents of CS and EIP registers point to the instruction following the instruction that generated the exception.

Program State Change

Fault — A program-state change does not accompany the debug exception, because the exception occurs before the faulting instruction is executed. The program can resume normal execution upon returning from the debug exception handler.

Trap — A program-state change does accompany the debug exception, because the instruction or task switch being executed is allowed to complete before the exception is generated. However, the new state of the program is not corrupted and execution of the program can continue reliably.

The following items detail the treatment of debug exceptions on the instruction boundary following execution of the MOV or the POP instruction that loads the SS register:

- If EFLAGS.TF is 1, no single-step trap is generated.
- If the instruction encounters a data breakpoint, the resulting debug exception is delivered after completion of the instruction after the MOV or POP. This occurs even if the next instruction is INT *n*, INT3, or INTO.
- Any instruction breakpoint on the instruction after the MOV or POP is suppressed (as if EFLAGS.RF were 1).

Any debug exception inside an RTM region causes a transactional abort and, by default, redirects control flow to the fallback instruction address. If advanced debugging of RTM transactional regions has been enabled, any transactional abort due to a debug exception instead causes execution to roll back to just before the XBEGIN instruction

and then delivers a #DB. See Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

Interrupt 2—NMI Interrupt

Exception Class **Not applicable.**

Description

The nonmaskable interrupt (NMI) is generated externally by asserting the processor's NMI pin or through an NMI request set by the I/O APIC to the local APIC. This interrupt causes the NMI interrupt handler to be called.

Exception Error Code

Not applicable.

Saved Instruction Pointer

The processor always takes an NMI interrupt on an instruction boundary. The saved contents of CS and EIP registers point to the next instruction to be executed at the point the interrupt is taken. See Section 7.5, "Exception Classifications," for more information about when the processor takes NMI interrupts.

Program State Change

The instruction executing when an NMI interrupt is received is completed before the NMI is generated. A program or task can thus be restarted upon returning from an interrupt handler without loss of continuity, provided the interrupt handler saves the state of the processor before handling the interrupt and restores the processor's state prior to a return.

Interrupt 3—Breakpoint Exception (#BP)

Exception Class **Trap.**

Description

Indicates that a breakpoint instruction (INT3, opcode CC) was executed, causing a breakpoint trap to be generated. Typically, a debugger sets a breakpoint by replacing the first opcode byte of an instruction with the opcode for the INT3 instruction. (The INT3 instruction is one byte long, which makes it easy to replace an opcode in a code segment in RAM with the breakpoint opcode.) The operating system or a debugging tool can use a data segment mapped to the same physical address space as the code segment to place an INT3 instruction in places where it is desired to call the debugger.

With the P6 family, Pentium, Intel486, and Intel386 processors, it is more convenient to set breakpoints with the debug registers. (See Section 20.3.2, “Breakpoint Exception (#BP)—Interrupt Vector 3,” for information about the breakpoint exception.) If more breakpoints are needed beyond what the debug registers allow, the INT3 instruction can be used.

Any breakpoint exception inside an RTM region causes a transactional abort and, by default, redirects control flow to the fallback instruction address. If advanced debugging of RTM transactional regions has been enabled, any transactional abort due to a break exception instead causes execution to roll back to just before the XBEGIN instruction and then delivers a **debug exception (#DB)** — **not** a breakpoint exception. See Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

A breakpoint exception can also be generated by executing the INT *n* instruction with an operand of 3. The action of this instruction (INT 3) is slightly different than that of the INT3 instruction (see “INT *n*/INT0/INT3/INT1—Call to Interrupt Procedure” in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A).

Exception Error Code

None.

Saved Instruction Pointer

Saved contents of CS and EIP registers point to the instruction following the INT3 instruction.

Program State Change

Even though the EIP points to the instruction following the breakpoint instruction, the state of the program is essentially unchanged because the INT3 instruction does not affect any register or memory locations. The debugger can thus resume the suspended program by replacing the INT3 instruction that caused the breakpoint with the original opcode and decrementing the saved contents of the EIP register. Upon returning from the debugger, program execution resumes with the replaced instruction.

Interrupt 4—Overflow Exception (#OF)

Exception Class **Trap.**

Description

Indicates that an overflow trap occurred when an INTO instruction was executed. The INTO instruction checks the state of the OF flag in the EFLAGS register. If the OF flag is set, an overflow trap is generated.

Some arithmetic instructions (such as the ADD and SUB) perform both signed and unsigned arithmetic. These instructions set the OF and CF flags in the EFLAGS register to indicate signed overflow and unsigned overflow, respectively. When performing arithmetic on signed operands, the OF flag can be tested directly or the INTO instruction can be used. The benefit of using the INTO instruction is that if the overflow exception is detected, an exception handler can be called automatically to handle the overflow condition.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction following the INTO instruction.

Program State Change

Even though the EIP points to the instruction following the INTO instruction, the state of the program is essentially unchanged because the INTO instruction does not affect any register or memory locations. The program can thus resume normal execution upon returning from the overflow exception handler.

Interrupt 5—BOUND Range Exceeded Exception (#BR)

Exception Class **Fault.**

Description

Indicates that a BOUND-range-exceeded fault occurred when a BOUND instruction was executed. The BOUND instruction checks that a signed array index is within the upper and lower bounds of an array located in memory. If the array index is not within the bounds of the array, a BOUND-range-exceeded fault is generated.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the BOUND instruction that generated the exception.

Program State Change

A program-state change does not accompany the bounds-check fault, because the operands for the BOUND instruction are not modified. Returning from the BOUND-range-exceeded exception handler causes the BOUND instruction to be restarted.

Interrupt 6—Invalid Opcode Exception (#UD)

Exception Class **Fault.**

Description

Indicates that the processor did one of the following things:

- Attempted to execute an invalid or reserved opcode.
- Attempted to execute an instruction with an operand type that is invalid for its accompanying opcode; for example, the source operand for a LES instruction is not a memory location.
- Attempted to execute an MMX or SSE/SSE2/SSE3 instruction on an Intel 64 or IA-32 processor that does not support the MMX technology or SSE/SSE2/SSE3/SSSE3 extensions, respectively. CPUID feature flags MMX (bit 23), SSE (bit 25), SSE2 (bit 26), SSE3 (ECX, bit 0), SSSE3 (ECX, bit 9) indicate support for these extensions.
- Attempted to execute an MMX instruction or SSE/SSE2/SSE3/SSSE3 SIMD instruction (with the exception of the MOVNTI, PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, CLFLUSH, MONITOR, and MWAIT instructions) when the EM flag in control register CR0 is set (1).
- Attempted to execute an SSE/SE2/SSE3/SSSE3 instruction when the OSFXSR bit in control register CR4 is clear (0). Note this does not include the following SSE/SSE2/SSE3 instructions: MASKMOVQ, MOVNTQ, MOVNTI, PREFETCHh, SFENCE, LFENCE, MFENCE, and CLFLUSH; or the 64-bit versions of the PAVGB, PAVGW, PEXTRW, PINSRW, PMAXSW, PMAXUB, PMINSW, PMINUB, PMOVMSKB, PMULHUW, PSADB, PSHUFW, PADDQ, PSUBQ, PALIGNR, PABSB, PABSD, PABSW, PHADDD, PHADDSW, PHADDW, PHSUBD, PHSUBSW, PHSUBW, PMADDUSB, PMULHRW, PSHUFB, PSIGNB, PSIGND, and PSIGNW.
- Attempted to execute an SSE/SSE2/SSE3/SSSE3 instruction on an Intel 64 or IA-32 processor that caused a SIMD floating-point exception when the OSXMMEXCPT bit in control register CR4 is clear (0).
- Executed a UD0, UD1, UD2, or UDB instruction. Note that even though it is the execution of the UD0, UD1, UD2, or UDB instruction that causes the invalid opcode exception, the saved instruction pointer will reference the UD0, UD1, UD2, or UDB instruction.
- Detected a LOCK prefix that precedes an instruction that may not be locked or one that may be locked but the destination operand is not a memory location.
- Attempted to execute an LLDT, SLDT, LTR, STR, LSL, LAR, VERR, VERW, or ARPL instruction while in real-address or virtual-8086 mode.
- Attempted to execute the RSM instruction when not in SMM mode.

In Intel 64 and IA-32 processors that implement out-of-order execution microarchitectures, this exception is not generated until an attempt is made to retire the result of executing an invalid instruction; that is, decoding and speculatively attempting to execute an invalid opcode does not generate this exception. Likewise, in the Pentium processor and earlier IA-32 processors, this exception is not generated as the result of prefetching and preliminary decoding of an invalid instruction. (See Section 7.5, “Exception Classifications,” for general rules for taking of interrupts and exceptions.)

The opcodes D6 and F1 are undefined opcodes reserved by the Intel 64 and IA-32 architectures. These opcodes, even though undefined, do not generate an invalid opcode exception.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

A program-state change does not accompany an invalid-opcode fault, because the invalid instruction is not executed.

Interrupt 7—Device Not Available Exception (#NM)

Exception Class **Fault.**

Description

Indicates one of the following things:

The device-not-available exception is generated by either of three conditions:

- The processor executed an x87 FPU floating-point instruction while the EM flag in control register CR0 was set (1). See the paragraph below for the special case of the WAIT/FWAIT instruction.
- The processor executed a WAIT/FWAIT instruction while the MP and TS flags of register CR0 were set, regardless of the setting of the EM flag.
- The processor executed an x87 FPU, MMX, or SSE/SSE2/SSE3 instruction (with the exception of MOVNTI, PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, and CLFLUSH) while the TS flag in control register CR0 was set and the EM flag is clear.

The EM flag is set when the processor does not have an internal x87 FPU floating-point unit. A device-not-available exception is then generated each time an x87 FPU floating-point instruction is encountered, allowing an exception handler to call floating-point instruction emulation routines.

The TS flag indicates that a context switch (task switch) has occurred since the last time an x87 floating-point, MMX, or SSE/SSE2/SSE3 instruction was executed; but that the context of the x87 FPU, XMM, and MXCSR registers were not saved. When the TS flag is set and the EM flag is clear, the processor generates a device-not-available exception each time an x87 floating-point, MMX, or SSE/SSE2/SSE3 instruction is encountered (with the exception of the instructions listed above). The exception handler can then save the context of the x87 FPU, XMM, and MXCSR registers before it executes the instruction. See Section 2.5, “Control Registers,” for more information about the TS flag.

The MP flag in control register CR0 is used along with the TS flag to determine if WAIT or FWAIT instructions should generate a device-not-available exception. It extends the function of the TS flag to the WAIT and FWAIT instructions, giving the exception handler an opportunity to save the context of the x87 FPU before the WAIT or FWAIT instruction is executed. The MP flag is provided primarily for use with the Intel 286 and Intel386 DX processors. For programs running on the Pentium 4, Intel Xeon, P6 family, Pentium, or Intel486 DX processors, or the Intel 487 SX coprocessors, the MP flag should always be set; for programs running on the Intel486 SX processor, the MP flag should be clear.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the floating-point instruction or the WAIT/FWAIT instruction that generated the exception.

Program State Change

A program-state change does not accompany a device-not-available fault, because the instruction that generated the exception is not executed.

If the EM flag is set, the exception handler can then read the floating-point instruction pointed to by the EIP and call the appropriate emulation routine.

If the MP and TS flags are set or the TS flag alone is set, the exception handler can save the context of the x87 FPU, clear the TS flag, and continue execution at the interrupted floating-point or WAIT/FWAIT instruction.

Interrupt 8—Double Fault Exception (#DF)

Exception Class **Abort.**

Description

Indicates that the processor detected a second exception while calling an exception handler for a prior exception. Normally, when the processor detects another exception while trying to call an exception handler, the two exceptions can be handled serially. If, however, the processor cannot handle them serially, it signals the double-fault exception. To determine when two faults need to be signalled as a double fault, the processor divides the exceptions into three classes: benign exceptions, contributory exceptions, and page faults (see Table 7-4).

Table 7-4. Interrupt and Exception Classes

Class	Vector Number	Description
Benign Exceptions and Interrupts	1	Debug
	2	NMI Interrupt
	3	Breakpoint
	4	Overflow
	5	BOUND Range Exceeded
	6	Invalid Opcode
	7	Device Not Available
	9	Coprocessor Segment Overrun
	16	Floating-Point Error
	17	Alignment Check
	18	Machine Check
	19	SIMD floating-point
	All	INT <i>n</i>
	All	INTR
Contributory Exceptions	0	Divide Error
	10	Invalid TSS
	11	Segment Not Present
	12	Stack Fault
	13	General Protection
	21	Control Protection
Page Faults	14	Page Fault
	20	Virtualization Exception

Table 7-5 shows the various combinations of exception classes that cause a double fault to be generated. A double-fault exception falls in the abort class of exceptions. The program or task cannot be restarted or resumed. The double-fault handler can be used to collect diagnostic information about the state of the machine and/or, when possible, to shut the application and/or system down gracefully or restart the system.

A segment or page fault may be encountered while prefetching instructions; however, this behavior is outside the domain of Table 7-5. Any further faults generated while the processor is attempting to transfer control to the appropriate fault handler could still lead to a double-fault sequence.

Table 7-5. Conditions for Generating a Double Fault

First Exception	Second Exception		
	Benign	Contributory	Page Fault
Benign	Handle Exceptions Serially	Handle Exceptions Serially	Handle Exceptions Serially
Contributory	Handle Exceptions Serially	Generate a Double Fault	Handle Exceptions Serially
Page Fault	Handle Exceptions Serially	Generate a Double Fault	Generate a Double Fault
Double Fault	Handle Exceptions Serially	Enter Shutdown Mode	Enter Shutdown Mode

If another contributory or page fault exception occurs while attempting to call the double-fault handler, the processor enters shutdown mode. This mode is similar to the state following execution of an HLT instruction. In this mode, the processor stops executing instructions until an NMI interrupt, SMI interrupt, hardware reset, or INIT# is received. The processor generates a special bus cycle to indicate that it has entered shutdown mode. Software designers may need to be aware of the response of hardware when it goes into shutdown mode. For example, hardware may turn on an indicator light on the front panel, generate an NMI interrupt to record diagnostic information, invoke reset initialization, generate an INIT initialization, or generate an SMI. If any events are pending during shutdown, they will be handled after an wake event from shutdown is processed (for example, A20M# interrupts).

If a shutdown occurs while the processor is executing an NMI interrupt handler, then only a hardware reset can restart the processor. Likewise, if the shutdown occurs while executing in SMM, a hardware reset must be used to restart the processor.

Exception Error Code

Zero. The processor always pushes an error code of 0 onto the stack of the double-fault handler.

Saved Instruction Pointer

The saved contents of CS and EIP registers are undefined.

Program State Change

A program-state following a double-fault exception is undefined. The program or task cannot be resumed or restarted. The only available action of the double-fault exception handler is to collect all possible context information for use in diagnostics and then close the application and/or shut down or reset the processor.

If the double fault occurs when any portion of the exception handling machine state is corrupted, the handler cannot be invoked and the processor must be reset.

Interrupt 9—Coproprocessor Segment Overrun

Exception Class **Abort. (Intel reserved; do not use. Recent IA-32 processors do not generate this exception.)**

Description

Indicates that an Intel386 CPU-based systems with an Intel 387 math coprocessor detected a page or segment violation while transferring the middle portion of an Intel 387 math coprocessor operand. The P6 family, Pentium, and Intel486 processors do not generate this exception; instead, this condition is detected with a general protection exception (#GP), interrupt 13.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

A program-state following a coprocessor segment-overrun exception is undefined. The program or task cannot be resumed or restarted. The only available action of the exception handler is to save the instruction pointer and reinitialize the x87 FPU using the FNINIT instruction.

Interrupt 10—Invalid TSS Exception (#TS)

Exception Class **Fault.**

Description

Indicates that there was an error related to a TSS. Such an error might be detected during a task switch or during the execution of instructions that use information from a TSS. Table 7-6 shows the conditions that cause an invalid TSS exception to be generated.

Table 7-6. Invalid TSS Conditions

Error Code Index	Invalid Condition
TSS segment selector index	The TSS segment limit is less than 67H for 32-bit TSS or less than 2CH for 16-bit TSS.
TSS segment selector index	During an IRET task switch, the TI flag in the TSS segment selector indicates the LDT.
TSS segment selector index	During an IRET task switch, the TSS segment selector exceeds descriptor table limit.
TSS segment selector index	During an IRET task switch, the busy flag in the TSS descriptor indicates an inactive task.
TSS segment selector index	During a task switch, an attempt to access data in a TSS results in a limit violation or canonical fault.
TSS segment selector index	During an IRET task switch, the backlink is a NULL selector.
TSS segment selector index	During an IRET task switch, the backlink points to a descriptor which is not a busy TSS.
TSS segment selector index	The new TSS descriptor is beyond the GDT limit.
TSS segment selector index	The new TSS selector is null on an attempt to lock the new TSS.
TSS segment selector index	The new TSS selector has the TI bit set on an attempt to lock the new TSS.
TSS segment selector index	The new TSS descriptor is not an available TSS descriptor on an attempt to lock the new TSS.
LDT segment selector index	LDT not valid or not present.
Stack segment selector index	The stack segment selector exceeds descriptor table limit.
Stack segment selector index	The stack segment selector is NULL.
Stack segment selector index	The stack segment descriptor is a non-data segment.
Stack segment selector index	The stack segment is not writable.
Stack segment selector index	The stack segment DPL $\geq$ CPL.
Stack segment selector index	The stack segment selector RPL $\geq$ CPL.
Code segment selector index	The code segment selector exceeds descriptor table limit.
Code segment selector index	The code segment selector is NULL.
Code segment selector index	The code segment descriptor is not a code segment type.
Code segment selector index	The nonconforming code segment DPL $\geq$ CPL.
Code segment selector index	The conforming code segment DPL is greater than CPL.
Data segment selector index	The data segment selector exceeds the descriptor table limit.
Data segment selector index	The data segment descriptor is not a readable code or data type.
Data segment selector index	The data segment descriptor is a nonconforming code type and RPL $>$ DPL.
Data segment selector index	The data segment descriptor is a nonconforming code type and CPL $>$ DPL.
TSS segment selector index	The TSS segment descriptor/upper descriptor is beyond the GDT segment limit.
TSS segment selector index	The TSS segment descriptor is not an available TSS type.
TSS segment selector index	The TSS segment descriptor is an available 286 TSS type in IA-32e mode.

Table 7-6. Invalid TSS Conditions (Contd.)

Error Code Index	Invalid Condition
TSS segment selector index	The TSS segment upper descriptor is not the correct type.
TSS segment selector index	The TSS segment descriptor contains a non-canonical base.

This exception can be generated either in the context of the original task or in the context of the new task (see Section 10.3, “Task Switching”). Until the processor has completely verified the presence of the new TSS, the exception is generated in the context of the original task. Once the existence of the new TSS is verified, the task switch is considered complete. Any invalid-TSS conditions detected after this point are handled in the context of the new task. (A task switch is considered complete when the task register is loaded with the segment selector for the new TSS and, if the switch is due to a procedure call or interrupt, the previous task link field of the new TSS references the old TSS.)

The invalid-TSS handler must be a task called using a task gate. Handling this exception inside the faulting TSS context is not recommended because the processor state may not be consistent.

Exception Error Code

An error code containing the segment selector index for the segment descriptor that caused the violation is pushed onto the stack of the exception handler. If the EXT flag is set, it indicates that the exception was caused by an event external to the currently running program (for example, if an external interrupt handler using a task gate attempted a task switch to an invalid TSS).

Saved Instruction Pointer

If the exception condition was detected before the task switch was carried out, the saved contents of CS and EIP registers point to the instruction that invoked the task switch. If the exception condition was detected after the task switch was carried out, the saved contents of CS and EIP registers point to the first instruction of the new task.

Program State Change

The ability of the invalid-TSS handler to recover from the fault depends on the error condition that causes the fault. See Section 10.3, “Task Switching,” for more information on the task switch process and the possible recovery actions that can be taken.

If an invalid TSS exception occurs during a task switch, it can occur before or after the commit-to-new-task point. If it occurs before the commit point, no program state change occurs. If it occurs after the commit point (when the segment descriptor information for the new segment selectors have been loaded in the segment registers), the processor will load all the state information from the new TSS before it generates the exception. During a task switch, the processor first loads all the segment registers with segment selectors from the TSS, then checks their contents for validity. If an invalid TSS exception is discovered, the remaining segment registers are loaded but not checked for validity and therefore may not be usable for referencing memory. The invalid TSS handler should not rely on being able to use the segment selectors found in the CS, SS, DS, ES, FS, and GS registers without causing another exception. The exception handler should load all segment registers before trying to resume the new task; otherwise, general-protection exceptions (#GP) may result later under conditions that make diagnosis more difficult. The Intel recommended way of dealing with this situation is to use a task for the invalid TSS exception handler. The task switch back to the interrupted task from the invalid-TSS exception-handler task will then cause the processor to check the registers as it loads them from the TSS.

Interrupt 11—Segment Not Present (#NP)

Exception Class **Fault.**

Description

Indicates that the present flag of a segment or gate descriptor is clear. The processor can generate this exception during any of the following operations:

- While attempting to load CS, DS, ES, FS, or GS registers. [Detection of a not-present segment while loading the SS register causes a stack fault exception (#SS) to be generated.] This situation can occur while performing a task switch.
- While attempting to load the LDTR using an LLDT instruction. Detection of a not-present LDT while loading the LDTR during a task switch operation causes an invalid-TSS exception (#TS) to be generated.
- When executing the LTR instruction and the TSS is marked not present.
- While attempting to use a gate descriptor or TSS that is marked segment-not-present, but is otherwise valid.

An operating system typically uses the segment-not-present exception to implement virtual memory at the segment level. If the exception handler loads the segment and returns, the interrupted program or task resumes execution.

A not-present indication in a gate descriptor, however, does not indicate that a segment is not present (because gates do not correspond to segments). The operating system may use the present flag for gate descriptors to trigger exceptions of special significance to the operating system.

A contributory exception or page fault that subsequently referenced a not-present segment would cause a double fault (#DF) to be generated instead of #NP.

Exception Error Code

An error code containing the segment selector index for the segment descriptor that caused the violation is pushed onto the stack of the exception handler. If the EXT flag is set, it indicates that the exception resulted from either:

- an external event (NMI or INTR) that caused an interrupt, which subsequently referenced a not-present segment
- a benign exception that subsequently referenced a not-present segment

The IDT flag is set if the error code refers to an IDT entry. This occurs when the IDT entry for an interrupt being serviced references a not-present gate. Such an event could be generated by an INT instruction or a hardware interrupt.

Saved Instruction Pointer

The saved contents of CS and EIP registers normally point to the instruction that generated the exception. If the exception occurred while loading segment descriptors for the segment selectors in a new TSS, the CS and EIP registers point to the first instruction in the new task. If the exception occurred while accessing a gate descriptor, the CS and EIP registers point to the instruction that invoked the access (for example a CALL instruction that references a call gate).

Program State Change

If the segment-not-present exception occurs as the result of loading a register (CS, DS, SS, ES, FS, GS, or LDTR), a program-state change does accompany the exception because the register is not loaded. Recovery from this exception is possible by simply loading the missing segment into memory and setting the present flag in the segment descriptor.

If the segment-not-present exception occurs while accessing a gate descriptor, a program-state change does not accompany the exception. Recovery from this exception is possible merely by setting the present flag in the gate descriptor.

If a segment-not-present exception occurs during a task switch, it can occur before or after the commit-to-new-task point (see Section 10.3, "Task Switching"). If it occurs before the commit point, no program state change

occurs. If it occurs after the commit point, the processor will load all the state information from the new TSS (without performing any additional limit, present, or type checks) before it generates the exception. The segment-not-present exception handler should not rely on being able to use the segment selectors found in the CS, SS, DS, ES, FS, and GS registers without causing another exception. (See the Program State Change description for “Interrupt 10—Invalid TSS Exception (#TS)” in this chapter for additional information on how to handle this situation.)

Interrupt 12—Stack Fault Exception (#SS)

Exception Class **Fault.**

Description

Indicates that one of the following stack related conditions was detected:

- A limit violation is detected during an operation that refers to the SS register. Operations that can cause a limit violation include stack-oriented instructions such as POP, PUSH, CALL, RET, IRET, ENTER, and LEAVE, as well as other memory references which implicitly or explicitly use the SS register (for example, MOV AX, [BP+6] or MOV AX, SS:[EAX+6]). The ENTER instruction generates this exception when there is not enough stack space for allocating local variables.
- A not-present stack segment is detected when attempting to load the SS register. This violation can occur during the execution of a task switch, a CALL instruction to a different privilege level, a return to a different privilege level, an LSS instruction, or a MOV or POP instruction to the SS register.
- A canonical violation is detected in 64-bit mode during an operation that reference memory using the stack pointer register containing a non-canonical memory address.

Recovery from this fault is possible by either extending the limit of the stack segment (in the case of a limit violation) or loading the missing stack segment into memory (in the case of a not-present violation).

In the case of a canonical violation that was caused intentionally by software, recovery is possible by loading the correct canonical value into RSP. Otherwise, a canonical violation of the address in RSP likely reflects some register corruption in the software.

Exception Error Code

If the exception is caused by a not-present stack segment or by overflow of the new stack during an inter-privilege-level call, the error code contains a segment selector for the segment that caused the exception. Here, the exception handler can test the present flag in the segment descriptor pointed to by the segment selector to determine the cause of the exception. For a normal limit violation (on a stack segment already in use) the error code is set to 0.

Saved Instruction Pointer

The saved contents of CS and EIP registers generally point to the instruction that generated the exception. However, when the exception results from attempting to load a not-present stack segment during a task switch, the CS and EIP registers point to the first instruction of the new task.

Program State Change

A program-state change does not generally accompany a stack-fault exception, because the instruction that generated the fault is not executed. Here, the instruction can be restarted after the exception handler has corrected the stack fault condition.

If a stack fault occurs during a task switch, it occurs after the commit-to-new-task point (see Section 10.3, “Task Switching”). Here, the processor loads all the state information from the new TSS (without performing any additional limit, present, or type checks) before it generates the exception. The stack fault handler should thus not rely on being able to use the segment selectors found in the CS, SS, DS, ES, FS, and GS registers without causing another exception. The exception handler should check all segment registers before trying to resume the new task; otherwise, general protection faults may result later under conditions that are more difficult to diagnose. (See the Program State Change description for “Interrupt 10—Invalid TSS Exception (#TS)” in this chapter for additional information on how to handle this situation.)

Interrupt 13—General Protection Exception (#GP)

Exception Class **Fault.**

Description

Indicates that the processor detected one of a class of protection violations called “general-protection violations.” The conditions that cause this exception to be generated comprise all the protection violations that do not cause other exceptions to be generated (such as, invalid-TSS, segment-not-present, stack-fault, or page-fault exceptions). The following conditions cause general-protection exceptions to be generated:

- Exceeding the segment limit when accessing the CS, DS, ES, FS, or GS segments.
- Exceeding the segment limit when referencing a descriptor table (except during a task switch or a stack switch).
- Transferring execution to a segment that is not executable.
- Writing to a code segment or a read-only data segment.
- Reading from an execute-only code segment.
- Loading the SS register with a segment selector for a read-only segment (unless the selector comes from a TSS during a task switch, in which case an invalid-TSS exception occurs).
- Loading the SS, DS, ES, FS, or GS register with a segment selector for a system segment.
- Loading the DS, ES, FS, or GS register with a segment selector for an execute-only code segment.
- Loading the SS register with the segment selector of an executable segment or a null segment selector.
- Loading the CS register with a segment selector for a data segment or a null segment selector.
- Accessing memory using the DS, ES, FS, or GS register when it contains a null segment selector.
- Switching to a busy task during a call or jump to a TSS.
- Using a segment selector on a non-IRET task switch that points to a TSS descriptor in the current LDT. TSS descriptors can only reside in the GDT. This condition causes a #TS exception during an IRET task switch.
- Violating any of the privilege rules described in Chapter 6, “Protection.”
- Exceeding the instruction length limit of 15 bytes (this only can occur when redundant prefixes are placed before an instruction).
- Loading the CR0 register with a set PG flag (paging enabled) and a clear PE flag (protection disabled).
- Loading the CR0 register with a set NW flag and a clear CD flag.
- Referencing an entry in the IDT (following an interrupt or exception) that is not an interrupt, trap, or task gate.
- Attempting to access an interrupt or exception handler through an interrupt or trap gate from virtual-8086 mode when the handler’s code segment DPL is greater than 0.
- Attempting to write a 1 into a reserved bit of CR4.
- Attempting to execute a privileged instruction when the CPL is not equal to 0 (see Section 6.9, “Privileged Instructions,” for a list of privileged instructions).
- Attempting to execute SGDT, SIDT, SLDT, SMSW, or STR when CR4.UMIP = 1 and the CPL is not equal to 0.
- Writing to a reserved bit in an MSR.
- Accessing a gate that contains a null segment selector.
- Executing the INT *n* instruction when the CPL is greater than the DPL of the referenced interrupt, trap, or task gate.
- The segment selector in a call, interrupt, or trap gate does not point to a code segment.
- The segment selector operand in the LLDT instruction is a local type (TI flag is set) or does not point to a segment descriptor of the LDT type.
- The segment selector operand in the LTR instruction is local or points to a TSS that is not available.
- The target code-segment selector for a call, jump, or return is null.

- If the PAE and/or PSE flag in control register CR4 is set and the processor detects any reserved bits in a page-directory-pointer-table entry set to 1. These bits are checked during a write to control registers CR0, CR3, or CR4 that causes a reloading of the page-directory-pointer-table entry.
- Attempting to write a non-zero value into the reserved bits of the MXCSR register.
- Executing an SSE/SSE2/SSE3 instruction that attempts to access a 128-bit memory location that is not aligned on a 16-byte boundary when the instruction requires 16-byte alignment. This condition also applies to the stack segment.

A program or task can be restarted following any general-protection exception. If the exception occurs while attempting to call an interrupt handler, the interrupted program can be restartable, but the interrupt may be lost.

Exception Error Code

The processor pushes an error code onto the exception handler's stack. If the fault condition was detected while loading a segment descriptor, the error code contains a segment selector to or IDT vector number for the descriptor; otherwise, the error code is 0. The source of the selector in an error code may be any of the following:

- An operand of the instruction.
- A selector from a gate which is the operand of the instruction.
- A selector from a TSS involved in a task switch.
- IDT vector number.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

In general, a program-state change does not accompany a general-protection exception, because the invalid instruction or operation is not executed. An exception handler can be designed to correct all of the conditions that cause general-protection exceptions and restart the program or task without any loss of program continuity.

If a general-protection exception occurs during a task switch, it can occur before or after the commit-to-new-task point (see Section 10.3, "Task Switching"). If it occurs before the commit point, no program state change occurs. If it occurs after the commit point, the processor will load all the state information from the new TSS (without performing any additional limit, present, or type checks) before it generates the exception. The general-protection exception handler should thus not rely on being able to use the segment selectors found in the CS, SS, DS, ES, FS, and GS registers without causing another exception. (See the Program State Change description for "Interrupt 10—Invalid TSS Exception (#TS)" in this chapter for additional information on how to handle this situation.)

General Protection Exception in 64-bit Mode

The following conditions cause general-protection exceptions in 64-bit mode:

- If the memory address is in a non-canonical form.
- If a segment descriptor memory address is in non-canonical form.
- If the target offset in a destination operand of a call or jmp is in a non-canonical form.
- If a code segment or 64-bit call gate overlaps non-canonical space.
- If the code segment descriptor pointed to by the selector in the 64-bit gate doesn't have the L-bit set and the D-bit clear.
- If the EFLAGS.NT bit is set in IRET.
- If the stack segment selector of IRET is null when going back to compatibility mode.
- If the stack segment selector of IRET is null going back to CPL3 and 64-bit mode.
- If a null stack segment selector RPL of IRET is not equal to CPL going back to non-CPL3 and 64-bit mode.
- If the proposed new code segment descriptor of IRET has both the D-bit and the L-bit set.

- If the segment descriptor pointed to by the segment selector in the destination operand is a code segment and it has both the D-bit and the L-bit set.
- If the segment descriptor from a 64-bit call gate is in non-canonical space.
- If the DPL from a 64-bit call-gate is less than the CPL or than the RPL of the 64-bit call-gate.
- If the type field of the upper 64 bits of a 64-bit call gate is not 0.
- If an attempt is made to load a null selector in the SS register in compatibility mode.
- If an attempt is made to load null selector in the SS register in CPL3 and 64-bit mode.
- If an attempt is made to load a null selector in the SS register in non-CPL3 and 64-bit mode where RPL is not equal to CPL.
- If an attempt is made to clear CR0.PG while IA-32e mode is enabled.
- If an attempt is made to set a reserved bit in CR3, CR4 or CR8.

Interrupt 14—Page-Fault Exception (#PF)

Exception Class **Fault.**

Description

Indicates that, with paging enabled (the PG flag in the CR0 register is set), the processor detected one of the following conditions while using the page-translation mechanism to translate a linear address to a physical address:

- The P (present) flag in a page-directory or page-table entry needed for the address translation is clear, indicating that a page table or the page containing the operand is not present in physical memory.
- The procedure does not have sufficient privilege to access the indicated page (that is, a procedure running in user mode attempts to access a supervisor-mode page). If the SMAP flag is set in CR4, a page fault may also be triggered by code running in supervisor mode that tries to access data at a user-mode address. If either the PKE flag or the PKS flag is set in CR4, the protection-key rights registers may cause page faults on data accesses to linear addresses with certain protection keys.
- Code running in user mode attempts to write to a read-only page. If the WP flag is set in CR0, the page fault will also be triggered by code running in supervisor mode that tries to write to a read-only page.
- An instruction fetch to a linear address that translates to a physical address in a memory page with the execute-disable bit set (for information about the execute-disable bit, see Chapter 5, “Paging”). If the SMEP flag is set in CR4, a page fault will also be triggered by code running in supervisor mode that tries to fetch an instruction from a user-mode address.
- One or more reserved bits in paging-structure entry are set to 1. See description below of RSVD error code flag.
- A shadow-stack access is made to a page that is not a shadow-stack page. See Section 18.2, “Shadow Stacks,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, and Section 5.6, “Access Rights.”
- An enclave access violates one of the specified access-control requirements. See Section 38.3, “Access-control Requirements,” and Section 38.20, “Enclave Page Cache Map (EPCM),” in Chapter 38, “Enclave Access Control and Data Structures.” In this case, the exception is called an **SGX-induced page fault**. The processor uses the error code (below) to distinguish SGX-induced page faults from ordinary page faults.

The exception handler can recover from page-not-present conditions and restart the program or task without any loss of program continuity. It can also restart the program or task after a privilege violation, but the problem that caused the privilege violation may be uncorrectable.

See also: Section 5.7, “Page-Fault Exceptions.”

Exception Error Code

Yes (special format). The processor provides the page-fault handler with two items of information to aid in diagnosing the exception and recovering from it:

- An error code on the stack. The error code for a page fault has a format different from that for other exceptions (see Figure 7-11). The processor establishes the bits in the error code as follows:
 - P flag (bit 0).
This flag is 0 if there is no translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address.
 - W/R (bit 1).
If the access causing the page-fault exception was a write, this flag is 1; otherwise, it is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
 - U/S (bit 2).
If a user-mode access caused the page-fault exception, this flag is 1; it is 0 if a supervisor-mode access did so. This flag describes the access causing the page-fault exception, not the access rights specified by paging.

- RSVD flag (bit 3).
This flag is 1 if there is no translation for the linear address because a reserved bit was set in one of the paging-structure entries used to translate that address.
- I/D flag (bit 4).
This flag is 1 if the access causing the page-fault exception was an instruction fetch. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- PK flag (bit 5).
This flag is 1 if the access causing the page-fault exception was a data access to a linear address with a protection key for which the protection-key rights registers disallow access.
- SS (bit 6).
If the access causing the page-fault exception was a shadow-stack access (including shadow-stack accesses in enclave mode), this flag is 1; otherwise, it is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- HLAT (bit 7).
This flag is 1 if there is no translation for the linear address using HLAT paging because, in one of the paging-structure entries used to translate that address, either the P flag was 0 or a reserved bit was set. An error code will set this flag only if it clears bit 0 or sets bit 3. This flag will not be set by a page fault resulting from a violation of access rights, nor for one encountered during ordinary paging, including the case in which there has been a restart of HLAT paging.
- SGX flag (bit 15).
This flag is 1 if the exception is unrelated to paging and resulted from violation of SGX-specific access-control requirements. Because such a violation can occur only if there is no ordinary page fault, this flag is set only if the P flag (bit 0) is 1 and the RSVD flag (bit 3) and the PK flag (bit 5) are both 0.

See Section 5.6, “Access Rights,” and Section 5.7, “Page-Fault Exceptions,” for more information about page-fault exceptions and the error codes that they produce.

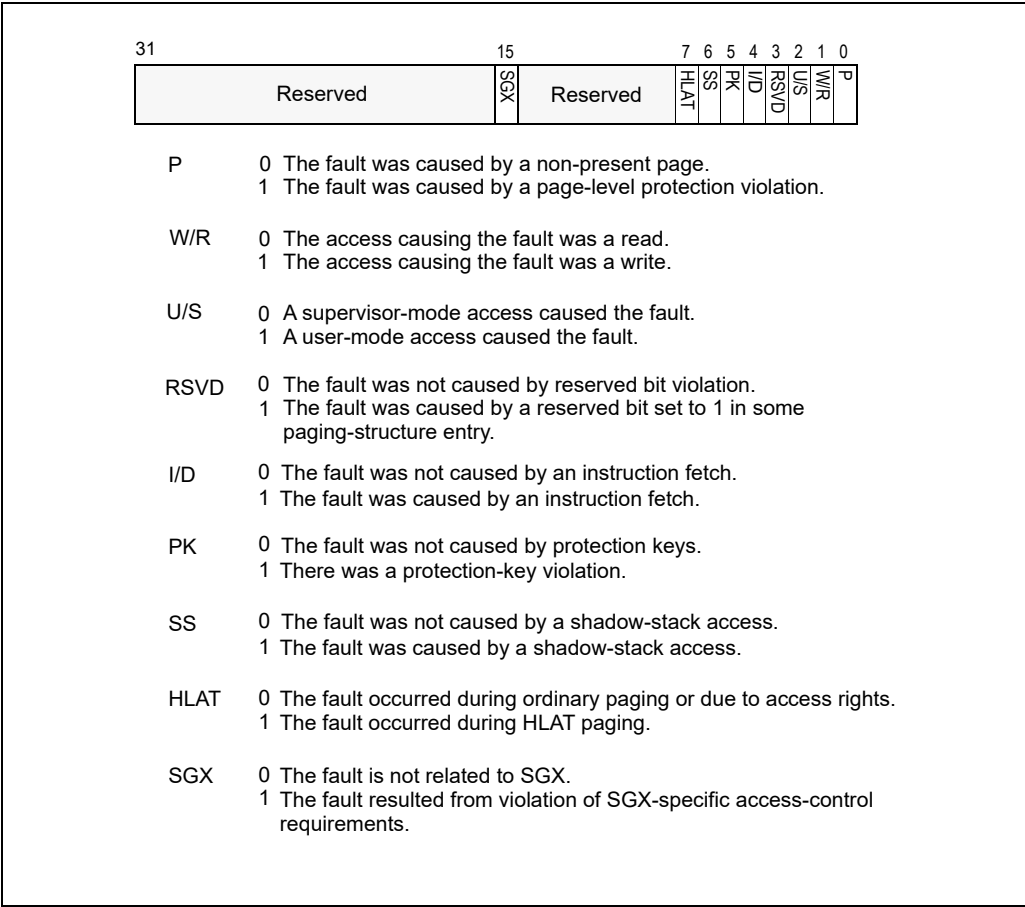


Figure 7-11. Page-Fault Error Code

- The contents of the CR2 register. The processor loads the CR2 register with the linear address that generated the exception. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata. If the page-fault exception occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the address are cleared.
The page-fault handler can use this address to locate the corresponding paging-structure entries. Another page fault can potentially occur during execution of the page-fault handler; the handler should save the contents of the CR2 register before a second page fault can occur.¹ If a page fault is caused by a page-level protection violation, the accessed flags in paging-structure entries may be set when the fault occurs (behavior is model-specific and not architecturally defined).

Saved Instruction Pointer

The saved contents of CS and EIP registers generally point to the instruction that generated the exception. If the page-fault exception occurred during a task switch, the CS and EIP registers may point to the first instruction of the new task (as described in the following “Program State Change” section).

1. Processors update CR2 whenever a page fault is detected. If a second page fault occurs while an earlier page fault is being delivered, the faulting linear address of the second fault will overwrite the contents of CR2 (replacing the previous address). These updates to CR2 occur even if the page fault results in a double fault or occurs during the delivery of a double fault.

Program State Change

A program-state change does not normally accompany a page-fault exception, because the instruction that causes the exception to be generated is not executed. After the page-fault exception handler has corrected the violation (for example, loaded the missing page into memory), execution of the program or task can be resumed.

When a page-fault exception is generated during a task switch, the program-state may change, as follows. During a task switch, a page-fault exception can occur during any of following operations:

- While writing the state of the original task into the TSS of that task.
- While reading the GDT to locate the TSS descriptor of the new task.
- While reading the TSS of the new task.
- While reading segment descriptors associated with segment selectors from the new task.
- While reading the LDT of the new task to verify the segment registers stored in the new TSS.

In the last two cases the exception occurs in the context of the new task. The instruction pointer refers to the first instruction of the new task, not to the instruction which caused the task switch (or the last instruction to be executed, in the case of an interrupt). If the design of the operating system permits page faults to occur during task-switches, the page-fault handler should be called through a task gate.

If a page fault occurs during a task switch, the processor will load all the state information from the new TSS (without performing any additional limit, present, or type checks) before it generates the exception. The page-fault handler should thus not rely on being able to use the segment selectors found in the CS, SS, DS, ES, FS, and GS registers without causing another exception. (See the Program State Change description for “Interrupt 10—Invalid TSS Exception (#TS)” in this chapter for additional information on how to handle this situation.)

Additional Exception-Handling Information

Special care should be taken to ensure that an exception that occurs during an explicit stack switch does not cause the processor to use an invalid stack pointer (SS:ESP). Software written for 16-bit IA-32 processors often use a pair of instructions to change to a new stack, for example:

```
MOV SS, AX
MOV SP, StackTop
```

When executing this code on one of the 32-bit IA-32 processors, it is possible to get a page fault, general-protection fault (#GP), or alignment check fault (#AC) after the segment selector has been loaded into the SS register but before the ESP register has been loaded. At this point, the two parts of the stack pointer (SS and ESP) are inconsistent. The new stack segment is being used with the old stack pointer.

The processor does not use the inconsistent stack pointer if the exception handler switches to a well defined stack (that is, the handler is a task or a more privileged procedure). However, if the exception handler is called at the same privilege level and from the same task, the processor will attempt to use the inconsistent stack pointer.

In systems that handle page-fault, general-protection, or alignment check exceptions within the faulting task (with trap or interrupt gates), software executing at the same privilege level as the exception handler should initialize a new stack by using the LSS instruction rather than a pair of MOV instructions, as described earlier in this note. When the exception handler is running at privilege level 0 (the normal case), the problem is limited to procedures or tasks that run at privilege level 0, typically the kernel of the operating system.

Interrupt 16—x87 FPU Floating-Point Error (#MF)

Exception Class **Fault.**

Description

Indicates that the x87 FPU has detected a floating-point error. The NE flag in the register CR0 must be set for an interrupt 16 (floating-point error exception) to be generated. (See Section 2.5, “Control Registers,” for a detailed description of the NE flag.)

NOTE

SIMD floating-point exceptions (#XM) are signaled through interrupt 19.

While executing x87 FPU instructions, the x87 FPU detects and reports six types of floating-point error conditions:

- Invalid operation (#I)
 - Stack overflow or underflow (#IS)
 - Invalid arithmetic operation (#IA)
- Divide-by-zero (#Z)
- Denormalized operand (#D)
- Numeric overflow (#O)
- Numeric underflow (#U)
- Inexact result (precision) (#P)

Each of these error conditions represents an x87 FPU exception type, and for each of exception type, the x87 FPU provides a flag in the x87 FPU status register and a mask bit in the x87 FPU control register. If the x87 FPU detects a floating-point error and the mask bit for the exception type is set, the x87 FPU handles the exception automatically by generating a predefined (default) response and continuing program execution. The default responses have been designed to provide a reasonable result for most floating-point applications.

If the mask for the exception is clear and the NE flag in register CR0 is set, the x87 FPU does the following:

1. Sets the necessary flag in the FPU status register.
2. Waits until the next “waiting” x87 FPU instruction or WAIT/FWAIT instruction is encountered in the program’s instruction stream.
3. Generates an internal error signal that cause the processor to generate a floating-point exception (#MF).

Prior to executing a waiting x87 FPU instruction or the WAIT/FWAIT instruction, the x87 FPU checks for pending x87 FPU floating-point exceptions (as described in step 2 above). Pending x87 FPU floating-point exceptions are ignored for “non-waiting” x87 FPU instructions, which include the FNINIT, FNCLEX, FNSTSW, FNSTSW AX, FNSTCW, FNSTENV, and FNSAVE instructions. Pending x87 FPU exceptions are also ignored when executing the state management instructions FXSAVE and FXRSTOR.

All of the x87 FPU floating-point error conditions can be recovered from. The x87 FPU floating-point-error exception handler can determine the error condition that caused the exception from the settings of the flags in the x87 FPU status word. See “Software Exception Handling” in Chapter 8 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for more information on handling x87 FPU floating-point exceptions.

Exception Error Code

None. The x87 FPU provides its own error information.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the floating-point or WAIT/FWAIT instruction that was about to be executed when the floating-point-error exception was generated. This is not the faulting instruction in which the error condition was detected. The address of the faulting instruction is contained in the x87 FPU instruction pointer

register. See Section 8.1.8, “x87 FPU Instruction and Data (Operand) Pointers,” in Chapter 8 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for more information about information the FPU saves for use in handling floating-point-error exceptions.

Program State Change

A program-state change generally accompanies an x87 FPU floating-point exception because the handling of the exception is delayed until the next waiting x87 FPU floating-point or WAIT/FWAIT instruction following the faulting instruction. The x87 FPU, however, saves sufficient information about the error condition to allow recovery from the error and re-execution of the faulting instruction if needed.

In situations where non- x87 FPU floating-point instructions depend on the results of an x87 FPU floating-point instruction, a WAIT or FWAIT instruction can be inserted in front of a dependent instruction to force a pending x87 FPU floating-point exception to be handled before the dependent instruction is executed. See “x87 FPU Exception Synchronization” in Chapter 8 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for more information about synchronization of x87 floating-point-error exceptions.

Interrupt 17—Alignment Check Exception (#AC)

Exception Class **Fault.**

Description

There are two causes of alignment-check exceptions: alignment violations and bus-lock violations.

Alignment Violations

An **alignment violation** occurs when the processor detects an unaligned memory operand when alignment checking was enabled. Alignment checks are only carried out in data (or stack) accesses (not in code fetches or system segment accesses). An example of an alignment violation is a word stored at an odd byte address, or a doubleword stored at an address that is not an integer multiple of 4. Table 7-7 lists the alignment requirements various data types recognized by the processor.

Table 7-7. Alignment Requirements by Data Type

Data Type	Address Must Be Divisible By
Word	2
Doubleword	4
Single precision floating-point (32-bits)	4
Double precision floating-point (64-bits)	8
Double extended precision floating-point (80-bits)	8
Quadword	8
Double quadword	16
Segment Selector	2
32-bit Far Pointer	2
48-bit Far Pointer	4
32-bit Pointer	4
GDTR, IDTR, LDTR, or Task Register Contents	4
FSTENV/FLDENV Save Area	4 or 2, depending on operand size
FSAVE/FRSTOR Save Area	4 or 2, depending on operand size
Bit String	2 or 4 depending on the operand-size attribute.

Note that an alignment violation occurs only for data types that must be aligned on word, doubleword, and quadword boundaries. A general-protection exception (#GP) is generated 128-bit data types that are not aligned on a 16-byte boundary.

To enable alignment checking, the following conditions must be true:

- AM flag in CR0 register is set.
- AC flag in the EFLAGS register is set.
- The CPL is 3 (including virtual-8086 mode).

Alignment violations are generated only when operating at privilege level 3 (user mode). Memory references that default to privilege level 0, such as segment descriptor loads, do not generate alignment violations, even when caused by a memory reference made from privilege level 3.

Storing the contents of the GDTR, IDTR, LDTR, or task register in memory while at privilege level 3 can generate an alignment violation. Although application programs do not normally store these registers, the fault can be avoided by aligning the information stored on an even word-address.

The FXSAVE/XSAVE and FXRSTOR/XRSTOR instructions save and restore a 512-byte data structure, the first byte of which must be aligned on a 16-byte boundary. If alignment violations are enabled when executing these instruc-

tions (and CPL is 3), a misaligned memory operand can cause either an alignment violation or a general-protection exception (#GP) depending on the processor implementation (see “FXSAVE—Save x87 FPU, MMX, SSE, and SSE2 State” and “FXRSTOR—Restore x87 FPU, MMX, SSE, and SSE2 State” in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A; see “XSAVE—Save Processor Extended States” and “XRSTOR—Restore Processor Extended States” in Chapter 6 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D).

The MOVDQU, MOVUPS, and MOVUPD instructions perform 128-bit unaligned loads or stores. The LDDQU instructions loads 128-bit unaligned data. They do not generate general-protection exceptions (#GP) when operands are not aligned on a 16-byte boundary. If alignment checking is enabled, alignment violations may or may not be generated depending on processor implementation when data addresses are not aligned on an 8-byte boundary.

FSAVE and FRSTOR instructions can generate unaligned references, which can cause alignment violations. These instructions are rarely needed by application programs.

Bus -Lock Violations

Some processors include features that disable bus locks. Section 11.1.2.3 provides details. When these features are enabled, occurrence of a bus lock causes a bus-lock violation, leading to a fault. In some cases, the fault is delivered as an alignment-check exception (#AC). The following are the cases in which bus-lock violation leads to an #AC:

- Split-lock disable is enabled (because MSR_MEMORY_CTRL[29] = 1) and locked access to multiple cache lines occurs (a split lock).
- Bus-lock disabled is enabled (because MSR_MEMORY_CTRL[28] = 1), CPUID.07H.02H:EDX[6] is enumerated as 1 (indicating support for the architectural form of bus-lock disable), and a locked access using a memory type other than WB occurs (a UC lock).¹

Exception Error Code

Yes. For alignment violations and bus-lock violations due to split locks, the error code is null, meaning that bits 31:2 of the error code are clear. For bus-lock violations due to UC locks, the error code has value 4, meaning that bit 2 is set.

In either case, bit 0 (EXT) is set if the violation is recognized during delivery of an event other than a software interrupt. (See Section 7.13, “Error Code” and “INT n/INTO/INT3/INT1—Call to Interrupt Procedure” in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.) In such cases, the actual error code delivered will have value 1 or value 5.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

A program-state change does not accompany an alignment-check fault, because the instruction is not executed.

1. If CPUID.07H.02H:EDX[6] is enumerated as 0, the processor may support an older model-specific form of bus-lock disable. That form generates a general-protection exception (#GP) and not an alignment-check exception.

Interrupt 18—Machine-Check Exception (#MC)

Exception Class **Abort.**

Description

Indicates that the processor detected an internal machine error or a bus error, or that an external agent detected a bus error. The machine-check exception is model-specific, available on the Pentium and later generations of processors. The implementation of the machine-check exception is different between different processor families, and these implementations may not be compatible with future Intel 64 or IA-32 processors. (Use the CPUID instruction to determine whether this feature is present.)

Bus errors detected by external agents are signaled to the processor on dedicated pins: the BINIT# and MCERR# pins on the Pentium 4, Intel Xeon, and P6 family processors and the BUSCHK# pin on the Pentium processor. When one of these pins is enabled, asserting the pin causes error information to be loaded into machine-check registers and a machine-check exception is generated.

The machine-check exception and machine-check architecture are discussed in detail in Chapter 18, “Machine-Check Architecture.” Also, see the data books for the individual processors for processor-specific hardware information.

Exception Error Code

None. Error information is provided by machine-check MSRs.

Saved Instruction Pointer

For the Pentium 4 and Intel Xeon processors, the saved contents of extended machine-check state registers are directly associated with the error that caused the machine-check exception to be generated (see Section 18.3.1.2, “IA32_MCG_STATUS MSR,” and Section 18.3.2.6, “IA32_MCG Extended Machine Check State MSRs”).

For the P6 family processors, if the EIPV flag in the MCG_STATUS MSR is set, the saved contents of CS and EIP registers are directly associated with the error that caused the machine-check exception to be generated; if the flag is clear, the saved instruction pointer may not be associated with the error (see Section 18.3.1.2, “IA32_MCG_STATUS MSR”).

For the Pentium processor, contents of the CS and EIP registers may not be associated with the error.

Program State Change

The machine-check mechanism is enabled by setting the MCE flag in control register CR4.

For the Pentium 4, Intel Xeon, P6 family, and Pentium processors, a program-state change always accompanies a machine-check exception, and an abort class exception is generated. For abort exceptions, information about the exception can be collected from the machine-check MSRs, but the program cannot generally be restarted.

If the machine-check mechanism is not enabled (the MCE flag in control register CR4 is clear), a machine-check exception causes the processor to enter the shutdown state.

Interrupt 19—SIMD Floating-Point Exception (#XM)

Exception Class **Fault.**

Description

Indicates the processor has detected an SSE/SSE2/SSE3 SIMD floating-point exception. The appropriate status flag in the MXCSR register must be set and the particular exception unmasked for this interrupt to be generated.

There are six classes of numeric exception conditions that can occur while executing an SSE/ SSE2/SSE3 SIMD floating-point instruction:

- Invalid operation (#I)
- Divide-by-zero (#Z)
- Denormal operand (#D)
- Numeric overflow (#O)
- Numeric underflow (#U)
- Inexact result (Precision) (#P)

The invalid operation, divide-by-zero, and denormal-operand exceptions are pre-computation exceptions; that is, they are detected before any arithmetic operation occurs. The numeric underflow, numeric overflow, and inexact result exceptions are post-computational exceptions.

See “SIMD Floating-Point Exceptions” in Chapter 11 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for additional information about the SIMD floating-point exception classes.

When a SIMD floating-point exception occurs, the processor does either of the following things:

- It handles the exception automatically by producing the most reasonable result and allowing program execution to continue undisturbed. This is the response to masked exceptions.
- It generates a SIMD floating-point exception, which in turn invokes a software exception handler. This is the response to unmasked exceptions.

Each of the six SIMD floating-point exception conditions has a corresponding flag bit and mask bit in the MXCSR register. If an exception is masked (the corresponding mask bit in the MXCSR register is set), the processor takes an appropriate automatic default action and continues with the computation. If the exception is unmasked (the corresponding mask bit is clear) and the operating system supports SIMD floating-point exceptions (the OSXM-MEXCPT flag in control register CR4 is set), a software exception handler is invoked through a SIMD floating-point exception. If the exception is unmasked and the OSXMMEXCPT bit is clear (indicating that the operating system does not support unmasked SIMD floating-point exceptions), an invalid opcode exception (#UD) is signaled instead of a SIMD floating-point exception.

Note that because SIMD floating-point exceptions are precise and occur immediately, the situation does not arise where an x87 FPU instruction, a WAIT/FWAIT instruction, or another SSE/SSE2/SSE3 instruction will catch a pending unmasked SIMD floating-point exception.

In situations where a SIMD floating-point exception occurred while the SIMD floating-point exceptions were masked (causing the corresponding exception flag to be set) and the SIMD floating-point exception was subsequently unmasked, then no exception is generated when the exception is unmasked.

When SSE/SSE2/SSE3 SIMD floating-point instructions operate on packed operands (made up of two or four sub-operands), multiple SIMD floating-point exception conditions may be detected. If no more than one exception condition is detected for one or more sets of sub-operands, the exception flags are set for each exception condition detected. For example, an invalid exception detected for one sub-operand will not prevent the reporting of a divide-by-zero exception for another sub-operand. However, when two or more exceptions conditions are generated for one sub-operand, only one exception condition is reported, according to the precedences shown in Table 7-8. This exception precedence sometimes results in the higher priority exception condition being reported and the lower priority exception conditions being ignored.

Table 7-8. SIMD Floating-Point Exceptions Priority

Priority	Description
1 (Highest)	Invalid operation exception due to SNaN operand (or any NaN operand for maximum, minimum, or certain compare and convert operations).
2	QNaN operand ¹ .
3	Any other invalid operation exception not mentioned above or a divide-by-zero exception ² .
4	Denormal operand exception ² .
5	Numeric overflow and underflow exceptions possibly in conjunction with the inexact result exception ² .
6 (Lowest)	Inexact result exception.

NOTES:

1. Though a QNaN this is not an exception, the handling of a QNaN operand has precedence over lower priority exceptions. For example, a QNaN divided by zero results in a QNaN, not a divide-by-zero- exception.
2. If masked, then instruction execution continues, and a lower priority exception can occur as well.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the SSE/SSE2/SSE3 instruction that was executed when the SIMD floating-point exception was generated. This is the faulting instruction in which the error condition was detected.

Program State Change

A program-state change does not accompany a SIMD floating-point exception because the handling of the exception is immediate unless the particular exception is masked. The available state information is often sufficient to allow recovery from the error and re-execution of the faulting instruction if needed.

Interrupt 20—Virtualization Exception (#VE)

Exception Class **Fault.**

Description

Indicates that the processor detected an EPT violation in VMX non-root operation. Not all EPT violations cause virtualization exceptions. See Section 28.5.7.2 for details.

The exception handler can recover from EPT violations and restart the program or task without any loss of program continuity. In some cases, however, the problem that caused the EPT violation may be uncorrectable.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers generally point to the instruction that generated the exception.

Program State Change

A program-state change does not normally accompany a virtualization exception, because the instruction that causes the exception to be generated is not executed. After the virtualization exception handler has corrected the violation (for example, by executing the EPTP-switching VM function), execution of the program or task can be resumed.

Additional Exception-Handling Information

The processor saves information about virtualization exceptions in the virtualization-exception information area. See Section 28.5.7.2 for details.

Interrupt 21—Control Protection Exception (#CP)

Exception Class **Fault.**

Description

Indicates a control flow transfer attempt violated the control flow enforcement technology constraints.

Exception Error Code

Yes (special format). The processor provides the control protection exception handler with following information through the error code on the stack.

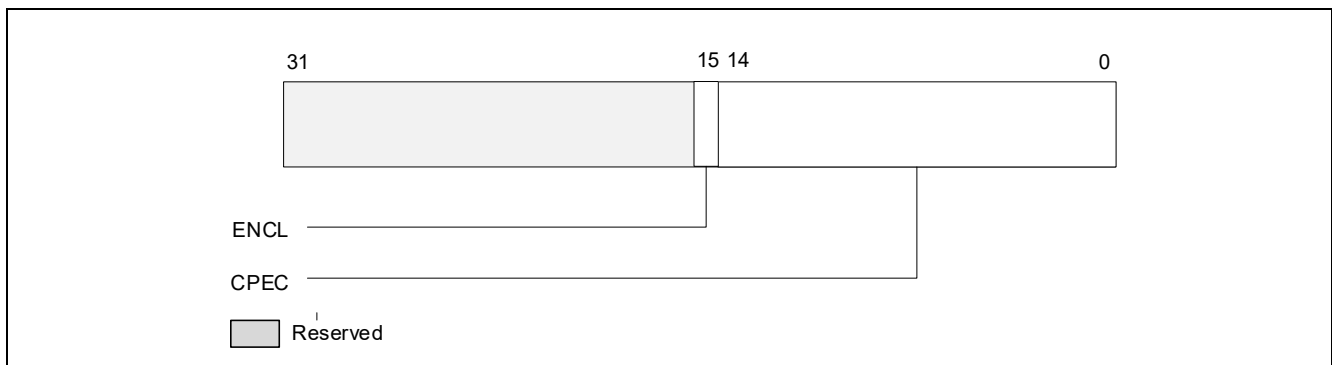


Figure 7-12. Exception Error Code Information

- Bit 14:0 - CPEC
 - 1 - NEAR-RET: Indicates the #CP was caused by a near RET instruction.
 - 2 - FAR-RET/IRET: Indicates the #CP was caused by a FAR RET or IRET instruction.
 - 3 - ENDBRANCH: indicates the #CP was due to missing ENDBRANCH at target of an indirect call or jump instruction.
 - 4 - RSTORSSP: Indicates the #CP was caused by a shadow-stack-restore token check failure in the RSTORSSP instruction.
 - 5- SETSSBSY: Indicates #CP was caused by a supervisor shadow stack token check failure in the SETSSBSY instruction.
- Bit 15 (ENCL) of the error code, if set to 1, indicates the #CP occurred during enclave execution.

Saved Instruction Pointer

The saved contents of the CS and EIP registers generally point to the instruction that generated the exception.

Program State Change

A program-state change does not normally accompany a control protection exception, because the instruction that causes the exception to be generated is not executed.

When a control protection exception is generated during a task switch, the program-state may change as follows. During a task switch, a control protection exception can occur during any of following operations:

- If task switch is initiated by IRET, CS and LIP stored on old task shadow stack do not match CS and LIP of new task (where LIP is the linear address of the return address).
- If task switch is initiated by IRET and SSP of new task loaded from shadow stack of old task (if new task CPL is < 3), OR the SSP from IA32_PL3_SSP (if new task CPL = 3) is not aligned to 4 bytes or is a value beyond 4GB.

In these cases the exception occurs in the context of the new task. The instruction pointer refers to the first instruction of the new task, not to the instruction which caused the task switch (or the last instruction to be executed, in the case of an interrupt). If the design of the operating system permits control protection faults to occur during task-switches, the control protection fault handler should be called through a task gate.

Interrupts 32 to 255—User Defined Interrupts

Exception Class **Not applicable.**

Description

Indicates that the processor did one of the following things:

- Executed an INT *n* instruction where the instruction operand is one of the vector numbers from 32 through 255.
- Responded to an interrupt request at the INTR pin or from the local APIC when the interrupt vector number associated with the request is from 32 through 255.

Exception Error Code

Not applicable.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that follows the INT *n* instruction or instruction following the instruction on which the INTR signal occurred.

Program State Change

A program-state change does not accompany interrupts generated by the INT *n* instruction or the INTR signal. The INT *n* instruction generates the interrupt within the instruction stream. When the processor receives an INTR signal, it commits all state changes for all previous instructions before it responds to the interrupt; so, program execution can resume upon returning from the interrupt handler.

14. Updates to Chapter 8, Volume 3A

Change bars and text show changes to Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- Inserted new chapter 8 for Flexible Return and Event Delivery (FRED) and re-numbered subsequent chapters.

CHAPTER 8

FLEXIBLE RETURN AND EVENT DELIVERY (FRED)

This chapter describes the architecture of **flexible return and event delivery (FRED)**.

The FRED architecture defines new transitions for invoking and returning from event handlers. These transitions are **FRED event delivery** and, for returning from events, two **FRED return instructions**.

FRED event delivery can change the current privilege level (CPL) from 3 to 0, but it is used also to deliver events that occur while CPL = 0. One FRED instruction (ERETU) effects a return that changes the CPL from 0 to 3, while the other (ERETS) returns while leaving the CPL at 0. Collectively, FRED event delivery and the FRED return instructions are **FRED transitions**.

The following is the organization of this chapter:

- Section 8.1 gives an overview of the FRED architecture.
- Section 8.2 gives details on enumeration and configuration of the FRED architecture, including the new state defined for it.
- Section 8.3 explains FRED event delivery.
- Section 8.4 presents the FRED return instructions.
- Section 8.5 describes how enabling FRED transitions changes the operation of existing instructions (including those use for system call and return).

8.1 OVERVIEW OF THE FRED ARCHITECTURE

The following items summarize the principle elements of the FRED architecture:

- **FRED event delivery.** Any event that would normally cause IDT event delivery (e.g., an interrupt or exception) instead establishes the new context with a process that does not use existing data structures such as the IDT. The SYSCALL and SYSENTER instructions also use FRED event delivery in place of their existing operation.
- **FRED return instructions.** Two return instructions, ERETS and ERETU, effect returns from event handling. ERETS (event return to supervisor) is used to return from events incident to CPL 0 and does not change CPL; ERETU (event return to user) returns to software operating with CPL 3.
- **GS segment management.** FRED transitions that change the CPL (FRED event delivery and ERETU) update the GS base address in a manner similar to the SWAPGS instruction. In addition, the LKGS instruction allows system software to manage the GS segment more flexibly.

When FRED transitions are enabled, they are the only transitions that change the CPL. Since FRED event delivery is always to CPL 0 and ERETU returns only to CPL 3, it is not possible for CPL to be 1 or 2 while FRED transitions are enabled. Enabling FRED transitions also prevents entry to compatibility mode with CPL 0.

The FRED architecture is defined for use by a 64-bit operating system. Except where noted, FRED event delivery and the FRED instructions (ERETS, ERETU, and LKGS) apply only in IA-32e mode (IA32_EFER.LMA = 1) and not in legacy protected mode (or real-address mode). The processor state defined by the FRED architecture is accessible with MSR-access instructions (e.g., RDMSR, WRMSR) regardless of mode.

The FRED architecture includes the concept of a **stack level**. The **current stack level** (CSL) is a value in the range 0–3 that a logical processor tracks when CPL = 0. FRED event delivery determines the stack level associated with the event being delivered and, if it is greater than the CSL (or if CPL had been 3), loads the stack pointer (RSP) from the **FRED RSP MSR** associated with the event's stack level. (If supervisor shadow stacks are enabled, the stack level applies also to the shadow-stack pointer, SSP, which would be loaded from a **FRED SSP MSR**.) The FRED return instruction ERETS restores the old stack level. (Details of the MSRs are given in Section 8.2.3.)

The existing shadow-stack architecture includes a token-management mechanism to ensure shadow-stack integrity when switching shadow stacks. This mechanism uses locked read-modify-write operations. FRED transitions do not use this token-management mechanism and instead protect shadow-stack integrity by having the FRED return instructions check values in the FRED SSP MSRs.

NOTE

Because FRED transitions do not use the shadow-stack token-management mechanism, FRED event delivery can never cause a fault or a VM exit after setting the busy bit in a shadow-stack token (unlike IDT event delivery). For that reason, an OS that enables FRED transitions need not consult CPUID.07H.1H:EDX.CET_SSS[18] before enabling supervisor shadow stacks.

8.2 ENUMERATION, ENABLING, AND CONFIGURATION

This section describes the enumeration, enabling, and configuration mechanisms for the FRED architecture.

8.2.1 Enumeration

The FRED architecture includes two related but independent elements: FRED transitions and the LKGS instruction for management of the GS segment register. Because 64-bit operating systems may benefit from the LKGS instruction without using FRED, the two elements are enumerated independently.

CPUID.07H.01H:EAX.FRED[17] is a feature flag that enumerates support for the FRED transitions. It also enumerates support for the architectural state (MSRs) defined by FRED.

CPUID.07H.01H:EAX.LKGS[18] is a feature flag that enumerates support for the LKGS instruction.

Any Intel® processor that enumerates support for FRED transitions will also enumerate support for LKGS.

8.2.2 Enabling in CR4

Software enables FRED transitions by setting CR4.FRED[32]. Specifically, setting CR4.FRED enables FRED event delivery (Section 8.3) as well as the FRED return instructions (Section 8.4), although those instructions can be used only in 64-bit mode (when CS.L = 1).

Execution of the MOV to CR4 instruction outside 64-bit mode clears CR4[63:32]. For that reason, software must first enter 64-bit mode before using MOV to CR4 to set CR4.FRED. The architecture ensures that CR4.FRED can be 1 only if the logical processor is in IA-32e mode (if IA32_EFER.LMA = 1):

- As just noted, MOV to CR4 can set CR4.FRED only in IA-32e mode.
- Software can leave IA-32e mode only by executing MOV to CR0 to clear CR0.PG, and this can be done only in compatibility mode while CPL = 0. (Attempts to clear CR0.PG in 64-bit mode will fault.)
- If CR4.FRED = 1, it is impossible to enter compatibility mode while CPL = 0 (see Section 8.5.2).
- RSM and VMX transitions cannot enter a state in which CR4.FRED = 1 outside IA-32e mode.

The value of CR4.FRED does not affect the accessibility of the FRED MSRs (Section 8.2.3) by MSR-access instructions (e.g., RDMSR, WRMSR), nor does it effect the operation of the LKGS instruction (Section 8.6).

Enabling FRED transitions changes the behavior of various existing instructions. See Section 8.5.

8.2.3 New MSRs for Configuration of FRED Transitions

FRED transitions are controlled by the following MSRs:

- IA32_FRED_CONFIG (MSR index 1D4H). This MSR is organized as follows:
 - Bits 1:0 of this MSR contain the **current stack level** (CSL). This 2-bit value is manipulated and used by FRED event delivery (Section 8.3) and the FRED return instructions (Section 8.4).
 - Software can modify the CSL with MSR-write instructions (e.g., WRMSR), but this is not an expected usage. It is recommended that, when using such an instruction to update IA32_FRED_CONFIG, software always write the existing CSL into bits 1:0 of the MSR (unless it specifically desires to change the CSL).
 - Bit 2 is reserved.

- Bit 3 indicates, if set, that, if FRED event delivery is not changing stacks, it should decrement the shadow-stack pointer (SSP) by 8.
- Bits 5:4 are reserved.
- Bits 8:6 identify the amount (measured in 64-byte cache lines) by which FRED event delivery decrements the regular stack pointer (RSP) when not changing stacks.
- Bits 10:9 identify the stack level that is used for maskable interrupts that occur while CPL = 0. See Section 8.3.1.2.
- Bit 11 is reserved.
- Bits 63:12 contain the upper bits of the linear address of a page in memory containing event handlers. FRED event delivery will load RIP to refer to an entry point on this page. See Section 8.3.1.1.

A write to this MSR (e.g., using WRMSR) causes a general-protection exception (#GP) if its operand sets any reserved bits or if it is not CPU canonical.

- IA32_FRED_RSP0, IA32_FRED_RSP1, IA32_FRED_RSP2, and IA32_FRED_RSP3 (MSR indices 1CCH–1CFH). These are the **FRED RSP MSRs**.

If FRED event delivery causes a change of the CPL (from 3 to 0) or a change to the CSL, it loads RSP from the FRED RSP MSR corresponding to the new stack level. See Section 8.3.1.2.

A write to any of these MSRs (e.g., using WRMSR) causes a general-protection exception (#GP) if its source operand is not 64-byte aligned (bits 5:0 are not all zero) or if it is not CPU canonical.

NOTE

The numbers 0–3 in the MSR names refer to the corresponding stack level and not to privilege level.

- IA32_FRED_STKLVL (MSR index 1D0H). This MSR is interpreted as an array of 32 2-bit values, one for each vector in the range 0–31. For an exception with vector v (or for a non-maskable interrupt, which has vector $v = 2$) that occurs CPL = 0, FRED event delivery ensures that the new stack level is at least the value of $\text{IA32\_FRED\_STKLVL}[2v+1:2v]$. See Section 8.3.1.2.
- IA32_FRED_SSP1, IA32_FRED_SSP2, and IA32_FRED_SSP3 (MSR indices 1D1H–1D3H). Together with the existing MSR IA32_PLO_SSP (MSR index 6A4H), these are the **FRED SSP MSRs**. References in this document to “IA32_FRED_SSP0” are to the IA32_PLO_SSP MSR.

If supervisor shadow stacks are enabled and FRED event delivery causes a change of the CPL (from 3 to 0) or a change to the CSL, it loads SSP from the FRED SSP MSR corresponding to the new stack level. See Section 8.3.1.2.

The address in each of IA32_FRED_SSPi ($1 \leq i \leq 3$) must be 8-byte aligned, so bits 2:0 are reserved. (IA32_PLO_SSP — also called IA32_FRED_SSP0 — is allowed to be 4-byte aligned, so only bits 1:0 are reserved.)

A write to any of these MSRs (e.g., using WRMSR) will cause a general-protection exception (#GP) if its source operand sets any reserved bits or if it is not CPU canonical. (The same is already true for executions of XRSTORS and MSR-write instructions that would load IA32_PLO_SSP.)

NOTES

As with the FRED RSP MSRs, the numbers 0–3 in the MSR names refer to the corresponding stack level **and not to privilege level**.

The FRED SSP MSRs are supported by any processor that enumerates CPUID.07H.01H:EAX.FRED[17] as 1. If such a processor does not support CET, FRED transitions will not use the MSRs (because shadow stacks are not enabled), but the MSRs would still be accessible using MSR-access instructions (e.g., RDMSR, WRMSR).

The RESET state of each of the new MSRs is zero. INIT does not change the value of the FRED MSRs.

8.2.4 FRED Use of Existing MSRs

In addition to the MSRs identified in Section 8.2.3, FRED transitions use the MSRs specified in the following items:

- IA32_STAR.
 - Use without FRED: this MSR is used by the existing operation of SYSCALL and SYSRET.
 - FRED use: FRED event delivery loads the CS and SS selectors with values derived from IA32_STAR[47:32] (see Section 8.3.3). ERETU uses the value of IA32_STAR[63:48] to determine how to load the CS and SS registers (see Section 8.4.2.1).

It is expected that operating systems will set IA32_STAR[33:32] to 00B and IA32_STAR[49:48] to 11B.
- IA32_KERNEL_GS_BASE.
 - Use without FRED: the SWAPGS instruction exchanges the value of this MSR with the base address of the GS segment register.
 - FRED use: executions of ERETU perform this same swapping operation, as does FRED event delivery of events that occur while CPL = 3.
- IA32_PLO_SSP.
 - Use without FRED: the shadow-stack feature of control-flow enforcement technology (CET) typically loads SSP from this MSR when the CPL changes to 0.
 - FRED use: FRED transitions use this MSR as IA32_FRED_SSP0 (Section 8.2.3).

8.3 FRED EVENT DELIVERY

When FRED transitions are enabled (CR4.FRED = 1), IDT event delivery of exceptions and interrupts is replaced with **FRED event delivery**. In addition, the existing operation of SYSCALL and SYSENTER is also replaced with FRED event delivery.

These changes do not affect the processor's handling of exceptions and interrupts prior to event delivery. For example, any determination that an event causes a VM exit or is converted into a double fault occurs normally. Similarly, page faults and debug exceptions update CR2 and DR6, respectively, in the normal way.

The principal functionality of FRED event delivery is to establish a new context, that of the event handler at CPL 0, while saving the old context for a subsequent return. Some parts of the new context have fixed values, while others depend on the old context, the nature of the event being delivered, and software configuration. Section 8.3.1 describes how FRED event delivery determines and establishes the new context. Section 8.3.2 specifies how FRED event delivery saves elements of the old context on the stack (and, when enabled, the shadow stack). Section 8.3.3 then describes how additional state is updated. Section 8.3.4 gives details about faults encountered during FRED event delivery.

Section 8.3.5 presents the detailed flow of the operation of FRED event delivery.

8.3.1 Determining and Establishing the New Context

The context of an event handler invoked by FRED event delivery includes the CS and SS segment registers, the instruction pointer (RIP), the flags register (RFLAGS), the stack pointer (RSP), and the base address of the GS segment (GS.base). The context also includes the shadow-stack pointer (SSP) if supervisor shadow stacks are enabled.

FRED event delivery establishes this context by loading these registers appropriately. It determines the values to be loaded into RIP, RSP, GS.base, and SSP based on the old context, the nature of the event being delivered, and software configuration. RFLAGS, CS, and SS are loaded with fixed values.

8.3.1.1 Determining the New RIP Value

FRED event delivery uses two entry points, depending on the CPL at the time the event occurred. This allows an event handler to identify the appropriate return instruction (ERETU or ERETS).

Specifically, the new RIP value that FRED event delivery establishes is `IA32_FRED_CONFIG & ~FFFH` for events that occur while `CPL = 3` and `(IA32_FRED_CONFIG & ~FFFH) + 256` for events that occur while `CPL = 0`.

If the new RIP value is not paging canonical, FRED event delivery causes a general-protection fault (`#GP`).¹

8.3.1.2 Determining the New Values for Stack Level, RSP, and SSP

FRED transitions support four different stacks for use while `CPL = 0`. A logical processor identifies the stack currently in use with a 2-bit value called the **current stack level (CSL)**.

FRED event delivery first determines the event's stack level and then uses that to determine whether the CSL should change. An event's stack level is based on the CPL, the nature and type of the event, the event's vector (for some event types), and MSRs configured by system software:

- If the event occurred while `CPL = 3`, was not a nested exception encountered during event delivery, and was not a double fault (`#DF`), the event's stack level is 0.
- If the event occurred while `CPL = 0`, was a nested exception encountered during event delivery, or was a `#DF`, the following items apply:
 - If the event is a maskable interrupt, the event's stack level is `IA32_FRED_CONFIG[10:9]`.
 - If the event is an exception or a non-maskable interrupt (NMI), the event's stack level is `IA32_FRED_STKLVL[2v+1:2v]`, where `v` is the event's vector (in the range 0–31).²
 - The stack level of all other events is 0.³

If the event occurred while `CPL = 3`, the new stack level is the event's stack level; otherwise, the new stack level is the maximum of the CSL and the event's stack level. (This implies that the CSL is always 0 following FRED event delivery of an event that occurred while `CPL = 3`, unless the event was a nested exception or a `#DF`.)

After determining the new stack level, FRED event delivery identifies the new RSP value as follows:

- If either the CPL or the CSL is changing, the new RSP value will be that of the FRED RSP MSR corresponding to the new stack level.
- Otherwise, the new RSP value will be the current RSP value decremented by `IA32_FRED_CONFIG & 1C0H` (the multiple of 64 in bits 8:6 of that MSR) and then aligned to a 64-byte boundary (by clearing `RSP[5:0]`).

If supervisor shadow stacks are enabled, the following items explain how the new SSP value is determined:

- If either the CPL or the CSL is changing, the new SSP value will be that of the FRED SSP MSR corresponding to the new stack level. If that new SSP value is not 8-byte aligned, FRED event delivery causes a general-protection fault (`#GP`).⁴
- Otherwise, the new SSP value will be the current SSP value decremented by `IA32_FRED_CONFIG & 8` (setting bit 3 of that MSR indicates that SSP should be decremented by 8).

8.3.1.3 Establishing the New Context

After determining the details of the new context, FRED event delivery loads registers to establish that context.

For events that occur while `CPL = 3`, FRED event delivery updates the CS, SS, and GS segments as well as the `IA32_KERNEL_GS_BASE` MSR:

- CS:
 - The selector is set to `IA32_STAR[47:32] AND FFFCH` (this forces `CS.RPL` to 0).

1. If this `#GP` does not cause a VM exit, FRED event delivery of the `#GP` will cause a second `#GP`, which will be converted to a double fault (`#DF`). If the `#DF` does not cause a VM exit, FRED event delivery of the `#DF` will cause a third `#GP`, resulting in a triple fault, which will either cause a VM exit or take the logical processor to shutdown.
2. Exceptions include the events generated by the `INT1`, `INT3`, and `INT0` instructions, but not those generated by the `INT n` instruction (for any value of `n`).
3. The other events are those generated by the instructions `INT n` (for any value of `n`), `SYSCALL`, and `SYSENTER`.
4. The value of each of `IA32_FRED_SSPI` ($1 \leq i \leq 3$) is always 8-byte aligned, so the check specified above is necessary only if SSP is being loaded from the `IA32_PLO_SSP` MSR (`IA32_FRED_SSPO`).

- The base address is set to 0. The limit is set to FFFFFFFH and the G bit is set to 1.
- The type is set to 11 (execute/read accessed code) and the S bit is set to 1.
- The DPL is set to 0, the P and L bits are each set to 1, and the D bit is set to 0.
- CS is made usable regardless of the value of the selector.
- SS:
 - The selector is set to the new value of the CS selector (above) plus 8 (thus, SS.RPL is also 0).
 - The base address is set to 0. The limit is set to FFFFFFFH and the G bit is set to 1.
 - The type is set to 3 (read/write accessed data) and the S bit is set to 1.
 - The DPL is set to 0, and the P and B bits are each set to 1.
 - SS is made usable regardless of the value of the selector.
- GS: FRED event delivery swaps the value of the GS base address and that of the IA32_KERNEL_GS_BASE MSR.

(Note that the above updates occur without any access to the GDT or an LDT.)

For events that occur while CPL = 0, FRED event delivery does not modify CS, SS, or GS.

After updating the segment registers, FRED event delivery loads RIP, RSP, and CSL with the values determined in Section 8.3.1.1 and Section 8.3.1.2. If supervisor shadow stacks are enabled, SSP is loaded with the value determined in Section 8.3.1.2. RFLAGS is loaded with the value 2 (all bits clear except position 1, which has no function but is always set).

If FRED event delivery incurs a nested exception or VM exit after this point, the processor restores the values that were in these registers before FRED event delivery commenced and only then delivers the nested exception or VM exit.

8.3.2 Saving Information About the Event and the Old Context

Like IDT event delivery, FRED event delivery saves information about the old context on the stack of the event handler. This information includes the previous context saved in much the same format as that following IDT event delivery, additional information about the event being delivered, and auxiliary information that will guide a subsequent return instruction.

If supervisor shadow stacks are enabled, FRED event delivery also saves information on the event handler's shadow stack.

Since FRED event delivery results in CPL = 0, the memory accesses used to store information on the stacks are performed with supervisor privilege. Any of these memory accesses may result in nested exception. The nested exception (or a double fault to which it may be converted) would then be delivered with FRED event delivery.

8.3.2.1 Saving Information on the Regular Stack

The format of the stack frame is illustrated in Figure 8-1 and detailed in the following items.

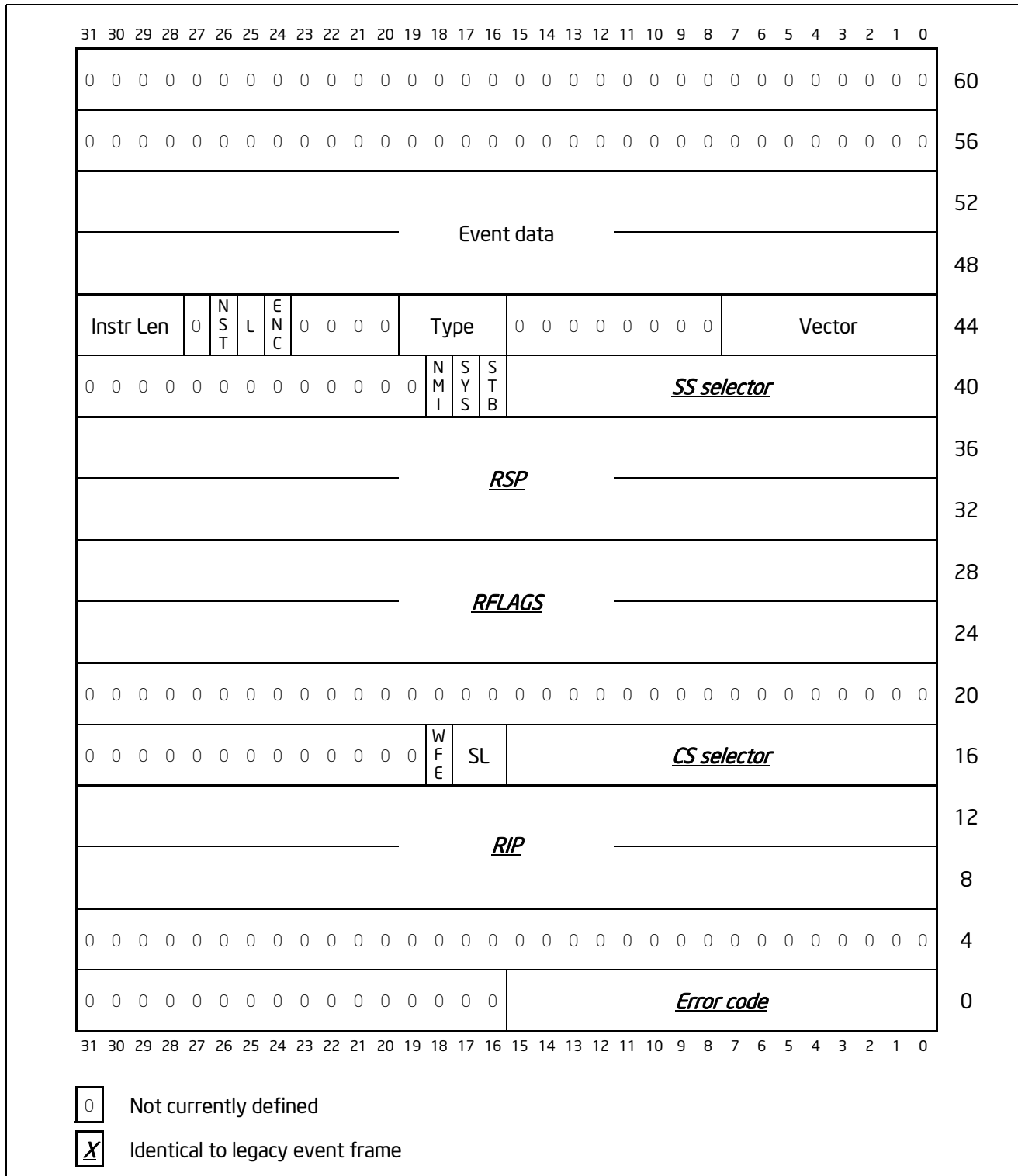


Figure 8-1. Flexible Return and Event Delivery Stack

- The first 8 bytes pushed (bytes 63:56 of the 64-byte stack frame) are not currently defined and will be zero until they are.

- The next 8 bytes pushed (bytes 55:48) contain **event data** and are defined as follows:
 - If the event being delivered is a page fault (#PF), the event data is the faulting linear address (this is the same value that the #PF loads into CR2).¹
 - If the event being delivered is a debug exception (#DB), the event data identifies the nature of the debug exception:
 - Bits 3:0 are B3–B0. When set, each of these bits indicates that the corresponding breakpoint condition was met (and may be set even if its corresponding enabling bit in DR7 is not set).
 - Bits 10:4 are not currently defined and will be zero until they are.
 - Bit 11 is BLD. When set, this bit indicates that the cause of the debug exception was acquisition of a bus lock (because IA32_DEBUGCTL[2] = 1).
 - Bit 12 is not currently defined and will be zero until it is.
 - Bit 13 is BD. When set, this bit indicates that the cause of the debug exception was “debug register access detected.”
 - Bit 14 is BS. When set, this bit indicates that the cause of the debug exception was the execution of a single instruction (because execution of that instruction commenced with RFLAGS.TF = 1).²
 - Bit 15 is not currently defined and will be zero until it is.
 - Bit 16 is RTM. When set, this bit indicates that a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled.
 - Bits 63:17 are not currently defined and will be zero until they are.

The event data is not exactly the same as that which will be in DR6 following delivery of the #DB. The polarity of bit 11 and bit 16 is inverted in DR6; in addition, delivery of #DB may leave unmodified in DR6 some bits in positions that are clear in the event data and that were already set in DR6.

- If the event being delivered is a device-not-available exception (#NM) caused by extended feature disable, the event data indicates the cause of the exception and is the same as that loaded into the IA32_XFD_ERR MSR. If the #NM was not caused by extended feature disable, the event data is zero.
- For any other event, the event data are not currently defined and will be zero until they are.
- The next 40 bytes pushed (bytes 47:8) are the **return state** and have generally the same format as that used by IDT event delivery. That format stores the 16-bit selectors for CS and SS in 64-bit fields. FRED event delivery augments those selector values with additional information saved in the upper 48 bits of those fields.

The following items detail the format of the return state on the stack from bottom (highest address) to top:

- The augmented SS of the previous context, 64 bits formatted as follows:
 - Bits 15:0 contain the SS selector.
 - Bit 16 is set to 1 for FRED event delivery of a hardware exception if interrupt blocking by STI (**STB**) was in effect at the time the exception occurred³ and is otherwise cleared to 0.⁴
 - Bit 17 is set to 1 for FRED event delivery of SYSCALL, SYSENTER, or INT n (**SYS**) and is otherwise cleared to 0.
 - Bit 18 is set to 1 if the event being delivered is a non-maskable interrupt (**NMI**) and is otherwise cleared to 0.
 - Bits 31:19 are not currently defined and will be zero until they are.

1. If the #PF occurred during execution of an instruction in enclave mode (but not during delivery of an event incident to enclave mode), bits 11:0 of the event data are cleared.
2. If in addition IA32_DEBUGCTL.BTF was 1 when the instruction commenced, a debug exception occurs only if the instruction was a taken branch.
3. Execution of STI with RFLAGS.IF = 0 blocks interrupts on the instruction boundary following its execution.
4. Delivery of SYSCALL, SYSENTER, INT1, INT3, or INT n (for any value of n) will always clear this bit, as will delivery of any exception encountered during event delivery for one of those instructions.

- Bits 63:32 contain **event information**. This provides software with additional information about the event. The FRED return instructions ignore these bits. The event information is formatted as follows:
 - Bits 39:32 contain the event's vector. The vector depends on the type of event:
 - For an external interrupt, the vector is provided by the local APIC.
 - For a non-maskable interrupt (NMI), the vector is 2.
 - For a hardware exception, the vector is determined by the exception.
 - For a software interrupt (INT n), the vector comes from the instruction's opcode (value n).
 - For a software exception, the vector is determined by the instruction opcode: INT1 has vector 1; INT3 has vector 3; INTO has vector 4.
 - For SYSCALL and SYSENTER (which use FRED event delivery but not IDT event delivery), vectors 1 and 2 are used, respectively.
 - Bits 47:40 are not currently defined and will be zero until they are.
 - Bit 51:48 encode the event type as follows: 0 = external interrupt; 2 = non-maskable interrupt; 3 = hardware exception (e.g., page fault); 4 = software interrupt (INT n); 5 = privileged software exception (INT1); 6 = software exception (INT3 or INTO); and 7 = other event (used for SYSCALL and SYSENTER). Other values are not used.¹
 - Bits 55:52 are not currently defined and will be zero until they are.
 - Bit 56 is set to 1 to indicate that the event was incident to enclave execution (**ENC**). Specifically, it is set if the event occurred while the logical processor was in enclave mode or was an exception encountered during delivery of such an event. Otherwise, the bit is cleared to 0.
 - Bit 57 (**L**) is set to 1 if the logical processor had been in 64-bit mode when the event occurred. (A value of 0 indicates an event incident to compatibility mode.)
 - Bit 58 is set to 1 if the event is nested exception (**NST**) encountered during FRED event delivery of another event. This bit is not set if the event is double fault (#DF).
 - Bit 59 is not currently defined and will be zero until it is.
 - Bits 63:60 contain the length of the instruction causing the event if the event type is software interrupt (INT n), privileged software exception (INT1), software exception (INT3 or INTO), or other event (when used for SYSCALL or SYSENTER). For other event types, these bits are cleared to zero.
- RSP of the previous context (64 bits).
- RFLAGS of the previous context (64 bits).

Bit 16 of the RFLAGS field (corresponding to the RF bit) is saved as 1 when delivering events that do the same for IDT event delivery. These are faults other than instruction breakpoints as well as any traps or interrupts delivered following partial execution of an instruction (e.g., between iterations of a REP-prefixed string instruction). Delivery of other events save in bit 16 the value that RFLAGS.RF had at the time the event occurred.
- The augmented CS of the previous context. The FRED return instructions use the 64 bits of this field. It is formatted as follows:
 - Bits 15:0 contain the CS selector.
 - Bits 17:16:
 - For delivery of events that occur while CPL = 0, these bits report the stack level (**SL**) at the time the event occurred.
 - For delivery of events that occur while CPL = 3, these bits are cleared to 0.
 - Bit 18 is set to 1 if the event occurred while CPL = 0 and the indirect branch tracker was in the WAIT_FOR_ENDBRANCH (**WFE**) state. Specifically, the bit is set if and only if the following are all true: CPL = 0; CR4.CET = 1; IA32_S_CET.ENDBR_EN = 1; and IA32_S_CET.TRACKER = 1.

1. The event types used by FRED event delivery are the same as those already defined for VMX transitions. For SYSCALL or SYSENTER, FRED event delivery reports event type 7 (other event).

The following items document the implications of how bit 18 is saved based on the existing operation of indirect branch tracking (providing details regarding delivery of specific events occurring while CPL = 0):

- Delivery of an event occurring at an instruction boundary saves bit 18 with the setting of the tracker at that instruction boundary:
 - For an event that takes priority over a missing-ENDBRANCH #CP exceptions, this value may be 0 or 1, depending on the state of the tracker. These events include certain machine-check exceptions, trap-class debug exceptions, non-maskable interrupts, external interrupts, debug exceptions due to instruction breakpoints, and faults from fetching the next instruction. They may also include faults from decoding the next instruction.
 - For a missing-ENDBRANCH #CP exception, this value is 1.
 - For a fault from decoding the next instruction that is prioritized below missing-ENDBRANCH #CP exceptions, this value is 0.
- Delivery of INT1 or INT3 saves bit 18 with the tracker's setting at the time INT1 or INT3 was fetched. (Decode of these instructions does not clear the tracker.)
- Delivery of INT n (for any value of n), SYSCALL, or SYSENTER will save bit 18 with value 0.
- Delivery of an exception encountered during FRED event delivery (including that for INT1, INT3, INT n, SYSCALL, or SYSENTER) will save bit 18 with the value that would have been saved by delivery of the original event.
- Delivery of an exception encountered during execution of any instruction other than INT1 or INT3 will save bit 18 with value 0.
- Bits 63:19 are not currently defined and will be zero until they are.
- RIP of the previous context (64 bits).

If the event type is software interrupt (INT n), privileged software exception (INT1), software exception (INT3 or INTO), or other event (when used for SYSCALL or SYSENTER); the RIP value saved references the instruction after the one that caused the event being delivered. (If delivery of such an event encounters an exception, the RIP value saved by delivery of the exception will instead reference the instruction that caused the original event.)

- The last 8 bytes pushed (bytes 7:0) contain error code (defined only for certain exceptions; zero if none is defined) in bits 15:0. Bits 63:16 are not currently defined and will be zero until they are.

8.3.2.2 Saving Information on the Shadow Stack

If supervisor shadow stacks are enabled and the event being delivered occurred while CPL = 0, FRED event delivery saves information on the new shadow stack. The following items provide details:

- FRED event delivery writes four bytes of zeroes to the linear address SSP – 4 and then clears SSP[2:0]. This has the following effect:
 - If the value of SSP had not been 8-byte aligned (it is necessarily 4-byte aligned), 4 bytes of zeroes are stored (as padding) and subsequent shadow-stack pushes will immediately follow those 4 bytes of zeroes.
 - If the value of SSP had been 8-byte aligned, 4 bytes of zeroes are stored but will be overwritten by subsequent shadow-stack pushes.
- FRED event delivery then pushes the following values on the new shadow stack:
 - The same 64-bit augmented CS that was pushed on the regular stack (see Section 8.3.2.1), including any bits defined in positions 63:16.
 - The old linear instruction pointer.
 - The old SSP on the shadow stack.

Each of these values is pushed in a separate 8-byte field on the shadow stack.

The write of zeroes and the pushes are all performed as shadow-stack accesses; because FRED event delivery results in CPL = 0, they are performed with supervisor privilege.

8.3.3 Loading Additional State

After saving the old context and other information, FRED event delivery completes operation by loading additional registers and updating other processor state.

If the event occurred while $CPL = 3$ and user shadow stacks are enabled, the `IA32_PL3_SSP` MSR is loaded with the old value of `SSP`. The value loaded into the MSR is adjusted so that bits 63:N get the value of bit N-1, where N is the processor's maximum linear-address width.

If supervisor indirect branch tracking is enabled, the `IA32_S_CET` MSR is updated to set the `TRACKER` value to `IDLE` and to clear the `SUPPRESS` bit to 0. There is no reason for the instruction referenced by the new `RIP` value to be `ENDBR64`.

FRED event delivery of a non-maskable interrupt (NMI) blocks NMIs.

A debug trap (single-step trap or data or I/O breakpoint) may be pending at the time another event is delivered. FRED event delivery drops any debug traps that were pending at the time the original event occurred, regardless of the event being delivered. In particular, any pending data or I/O breakpoints (or single-step traps) are no longer pending after `INT n`, `INT3`, `INTO`, `SYSCALL`, or `SYSENTER` is delivered using FRED event delivery.

Data breakpoints detected during FRED event delivery might not be dropped and may be pending after completion of FRED event delivery. A single-step trap will never be pending following FRED event delivery (even if `INT n`, `INT1`, `INT3`, `INTO`, `SYSCALL`, or `SYSENTER` is executed while `RFLAGS.TF = 1`).

8.3.4 Faults During FRED Event Delivery

The earlier sections identify situations in which faults may occur during FRED event delivery. In addition, a memory access will cause a fault if it uses a non-canonical address or encounters a LASS violation¹ (`#SS` for an access to the regular stack; `#GP` for the shadow stack); or if it causes a page fault (`#PF`).

An exception encountered during FRED event delivery will be converted to a double fault (`#DF`) following the same principles under which this is done for IDT event delivery. As with IDT event delivery, the error code for the `#DF` is always zero.

For a `#GP` or `#SS` encountered during FRED event delivery that is not converted to `#DF`, the error code will be 0 if the event that was being delivered had type software interrupt (`INT n`), software exception (`INT3` or `INTO`), or other event (`SYSCALL` and `SYSENTER`); it will be 1 for event types external interrupt, non-maskable interrupt, hardware exception, or privileged software exception (`INT1`). (This aligns with the existing specification of the `EXT` bit in such error codes.)

For a `#PF` encountered during FRED event delivery that is not converted to `#DF`, the error code is defined exactly as it is for IDT event delivery.

If FRED event delivery causes a fault, any changes to register state are undone before delivering the fault.

8.3.5 Detailed Operation of FRED Event Delivery

The section presents detailed pseudocode for the operation of FRED event delivery. It does not include details of how event information is formatted or how event data is determined. These details are in Section 8.3.2.1.

```
// HOLD OLD STATE IN TEMPORARIES
oldRIP := RIP;
oldCS := CS;                      // oldCS is 8 bytes; the upper 6 bytes are defined below
oldCPL := CPL;                    // the same as oldCS[1:0]
oldRFLAGS := RFLAGS;
oldSS := SS;                      // oldSS is 8 bytes; upper 6 bytes are zero
oldRSP := RSP;
IF CPL = 3
    THEN oldCSL := 0;
```

1. Since FRED event delivery clears both `CPL` and `RFLAGS.AC`, a LASS violation will occur if `CR4.LASS = CR4.SMAP = 1` and a memory access uses a linear address that clears bit 63.

```

    ELSE oldCSL := CSL;
FI;
oldGSB := GS.base;
oldSSP := SSP;                                // used only when shadow stacks are enabled
// ERETS and ERETU will restore the oldRFLAGS to RFLAGS
// Before saving, event delivery updates oldRFLAGS to set RF when appropriate
IF the event being delivered is a fault other than an instruction breakpoint OR
    the event being delivered is a trap or interrupt following partial execution of an instruction
    (e.g., between iterations of a REP-prefixed string instruction)
    THEN oldRFLAGS[16] := 1;
FI;
// Update bits above CS selector to hold additional information
// ERETS and ERETU will use these
oldCS[17:16] := oldCSL;
IF CPL = 3
    THEN oldCS[18] := 0;
    ELSE // See Section 8.3.2.1 for details of how oldCS[18] is determined
        oldCS[18] = CR4.CET AND IA32_S_CET.ENDBR_EN AND IA32_S_CET.TRACKER;
FI;
// Update bits above SS selector to hold additional information
IF STI blocking was in effect when the event occurred
    THEN oldSS[16] := 1;
FI;
IF event being delivered is either SYSCALL, SYSENTER, or INT n
    THEN oldSS[17] := 1;
FI;
IF event being delivered is an NMI                // includes VM-entry injection of NMI
    THEN oldSS[18] := 1;
FI;
// ERETS and ERETU will ignore the event information
oldSS[63:32] := event information as defined in Section 8.3.2.1;
// DETERMINE NEW CONTEXT
// determine new RIP
IF oldCPL = 3
    THEN newRIP := IA32_FRED_CONFIG & ~FFFH;
    ELSE newRIP := IA32_FRED_CONFIG & ~FFFH + 256;
FI;
IF newRIP is not paging canonical
    THEN #GP(0);                                // will lead eventually to shutdown or VM exit
FI;
// determine event's stack level and the new stack level for event handler
IF oldCPL = 3 AND event is not an exception nested on event delivery AND event is not #DF
    THEN eventSL := 0;
ELSE
    IF event type is external interrupt                // oldCPL must be 0
        THEN eventSL := IA32_FRED_CONFIG[10:9];
    ELSIF event type is hardware exception, software exception, privileged software exception, or NMI
        // v = event vector (0-31); includes INT1, INT3, INTO
        THEN eventSL := IA32_FRED_STKLVL[2v+1:2v];
        ELSE // SYSCALL, SYSENTER, INT n with CPL = 0; do not change CSL
            eventSL := 0;
FI;
FI;

```

```

newCSL := MAX[eventSL, oldCSL];
// determine new RSP
IF oldCPL = 3 OR newCSL > oldCSL
    THEN newRSP := IA32_FRED_RSPi, where i = newCSL;
    ELSE      // decrement RSP as specified and align to 64 bytes
        newRSP := (RSP - (IA32_FRED_CONFIG & 1COH)) & ~3FH;
FI;
// determine new RFLAGS; clear all bits except position 1
newRFLAGS := 2;
// if needed, determine new SSP
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1
    THEN
        IF oldCPL = 3 OR newCSL > oldCSL
            THEN
                newSSP := IA32_FRED_SSPi, where i = newCSL;
                IF newSSP[2] = 1      // check for 8-byte alignment
                    THEN #GP(0);
                FI;
            ELSE      // decrement SSP if specified
                newSSP := SSP - (IA32_FRED_CONFIG & 8);
        FI;
FI;

// ESTABLISH NEW CONTEXT — OLD STATE WILL BE RESTORED IF THERE IS A SUBSEQUENT EXCEPTION
// update segment registers if event occurred while CPL = 3
IF oldCPL = 3
    THEN
        // set CS to standard values used by a 64-bit operating system
        CS.selector := IA32_STAR[47:32] & FFFCH;
        CS.base := 0;
        CS.limit := FFFFFFFH;
        CS.type := 11;
        CS.S := 1;
        CS.DPL := 0;
        CS.P := 1;
        CS.L := 1;
        CS.D := 0;
        CS.G := 1;
        CS.unusable := 0;
        // set SS to standard values used by a 64-bit operating system
        SS.selector := CS.selector + 8;
        SS.base := 0;
        SS.limit := FFFFFFFH;
        SS.type := 3;
        SS.S := 1;
        SS.DPL := 0;
        SS.P := 1;
        SS.B := 1;
        SS.G := 1;
        SS.unusable := 0;
        // swap in supervisor GS base address
        GS.base := IA32_KERNEL_GS_BASE;
        IA32_KERNEL_GS_BASE := oldGSB;
FI;

```

```

// update registers defining context
RIP := newRIP;
RFLAGS := newRFLAGS;
RSP := newRSP;
CSL := newCSL;                                // reflected in IA32_FRED_CONFIG[1:0]
// Update SSP if supervisor shadow stacks enabled
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1
    THEN SSP := newSSP;
FI;

// SAVE STATE ON STACKS
// Save return state on new regular stack; memory accesses here have supervisor privilege
push8B 00000000_00000000H;                    // First 8 bytes pushed are all zero
push8B event data as defined in Section 8.3.2.1;
push8B oldSS;
push8B oldRSP;
push8B oldRFLAGS;
push8B oldCS;
push8B oldRIP;
push8B errorcode;

// If supervisor shadow stacks enabled and old CPL was 0, save data on new shadow stack
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1 AND oldCPL = 0
    THEN
        store 4 bytes of zeroes to address SSP - 4; // shadow-stack access
        SSP := SSP & ~7;                          // aligns SSP to 8B boundary
        pushSS_8B oldCS;                          // full 64-bit field
        pushSS_8B oldRIP;
        pushSS_8B oldSSP;
    FI;

// UPDATE ADDITIONAL STATE
// update additional CET state as needed (SSP was updated earlier)
IF CR4.CET = 1
    THEN
        IF oldCPL = 3 AND IA32_U_CET.SH_STK_EN = 1
            THEN // adjust so bits 63:N get the value of bit N-1
                    // N = processor's maximum linear-address width
                    IA32_PL3_SSP := LA_adjust(oldSSP);
            FI;
        IF IA32_S_CET.ENDBR_EN = 1
            THEN IA32_S_CET[11:10] := 00b;        // IDLE with SUPPRESS = 0
            FI;
    FI;

// update event-related state
clear any debug traps that were pending prior to FRED event delivery;
IF event being delivered is an NMI
    THEN block NMIs;
FI;
CPL := 0;

```

8.4 FRED RETURN INSTRUCTIONS

FRED defines two instructions for returning from events delivered by FRED event delivery:

- **ERETS** (Section 8.4.1) is used to return from events that occur while CPL = 0; it does not modify CS, SS, or GS.
- **ERETU** (Section 8.4.2) is used to return from events that occur while CPL = 3; it loads CS and SS based on the stack image and also updates the GS base address.

Neither instruction takes explicit operands. These instructions can be executed only if FRED transitions are enabled and only if CPL = 0; in addition, ERETU can be executed only if CSL = 0.

NOTE

The following sections indicate situations in which one of the instructions may cause a control-protection exception (#CP). These exceptions will use the value 2 for an error code, indicating that the #CP was caused by a return instruction other than near RET.

8.4.1 ERETU (Event Return to Supervisor)

ERETS returns from an event handler without changing the CPL, establishing the **return state** that was in effect before FRED event delivery. Because it stays within the supervisor context, ERETU does not modify the segment registers CS, SS, or GS. Execution of ERETU causes an invalid-opcode exception (#UD) if FRED transitions are not enabled or if CPL > 0.

ERETS begins by reading and checking the return state from the stack (Section 8.4.1.1). If supervisor shadow stacks are enabled, it then checks the shadow stack to confirm the validity of this control-flow transfer (Section 8.4.1.2). Finally, ERETU establishes the return state by loading the appropriate registers (Section 8.4.1.3).

See “ERETS—Event Return to Supervisor” in Chapter 3 of Volume 2 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual for the detailed flow of the operation of ERETU.

8.4.1.1 Loading and Checking the Return State

ERETS operates on the stack frame created by FRED event delivery (Section 8.3.2.1), principally to restore the state saved in that stack frame (the return state).

When FRED event delivery completes, the return state is located at offset 8 from the RSP value. (RSP references an 8-byte field containing the event’s error code.) For this reason, ERETU begins by adding 8 to RSP so that it can access the return state directly.

After adding 8 to RSP, ERETU pops from the regular stack (referenced by RSP) the return state that was saved by FRED event delivery. The state is checked and held by the processor to update register state when the instruction completes. The following items detail the state fields that are popped, from top (lowest address) to bottom (highest address), specifying the checks that are performed:

- The RIP of the return state (64 bits).
This field must be paging canonical; otherwise, a #GP with a zero error code occurs.
- The augmented CS for the return state (64 bits).
ERETS does not use bits 15:0 to load CS, but it does ensure that the value of these bits equals that of the current CS selector; otherwise, a #GP occurs.
Bit 18 of this field determines whether ERETU updates the state of the indirect branch tracker.
ERETS will establish the new stack level as the minimum of the CSL and the value of bits 17:16 of this field.
Bits 63:19 of this field must be zero; otherwise, a #GP occurs.
- The RFLAGS of the return state (64 bits).
Bit 1 of this field must be 1; bit 3, bit 5, bit 15, bit 17 (VM), and bits 63:22 of the field must be zero; otherwise, a #GP occurs.
- The RSP of the return state (64 bits). This field is not checked.
- The augmented SS of the return state (64 bits).
ERETS does not use bits 15:0 to load SS, but it does ensure that the value of these bits equals that of the current SS selector; otherwise, a #GP occurs.

Bits 18:16 of this field determine how ERETS manages certain events. See Section 8.4.1.3.

Bits 31:19 of this field must be zero; otherwise, a #GP occurs. ERETS ignores bits 63:32 of this field (the event information).

8.4.1.2 Checking the Shadow Stack

If supervisor shadow stacks are enabled, ERETS pops and checks values from the shadow stack (referenced by SSP) that were saved by FRED event delivery. The following items detail the state that is popped from top (lowest address) to bottom (highest address), specifying the checks that are performed:

- The SSP of the return state (64 bits).
This value must be 4-byte aligned (bits 1:0 must be zero); otherwise, a control-protection exception (#CP) occurs.
This value must be CPU canonical; otherwise, a #GP occurs with a zero error code. (This #GP has priority below the #CP exceptions specified in the following items.)
- The linear instruction pointer of the return state (64 bits).
This value must equal the RIP of the return context that was popped from the regular stack; otherwise, a #CP occurs.
- The augmented CS for the return state (64 bits).
This value must equal the code-segment value that was popped from the regular stack; otherwise, a #CP occurs.

If supervisor shadow stacks are enabled and the new stack level is less than the CSL, ERETS compares the value of SSP (after popping the above fields) and the value of the FRED SSP MSR for the CSL (not the new stack level). (For example, if ERETS is executing with CSL = 2 and is returning to stack level 1, the comparison is with IA32_FRED_SSP2.) If the values are not equal, a #CP occurs.

8.4.1.3 Establishing the Return State

After the stack operations described in Section 8.4.1.1 and Section 8.4.1.2, ERETS then loads RIP, RFLAGS, RSP, and CSL with the values that were popped earlier from the regular stack. If supervisor shadow stacks are enabled, SSP is loaded with the value that was popped earlier from the shadow stack.

Additional steps are performed based on the value of the popped fields for CS and SS:

- If bit 18 of the CS field is 1 and ERETS will complete with indirect branch tracking enabled and not suppressed, the indirect branch tracker will be in the WAIT_FOR_ENDBRANCH state upon completion of ERETS.¹
- If bit 16 of the SS field is 1, bit 9 of the RFLAGS field (corresponding to RFLAGS.IF) is 1, and there was no blocking by STI prior to ERETS, blocking by STI is in effect upon completion of ERETS.
- If bit 17 of the SS field is 1 and ERETS will result in RFLAGS.TF = 1, a single-step trap will be pending upon completion of ERETS.²
- If bit 18 of the SS field is 1, ERETS unblocks NMIs.

8.4.2 ERETU (Event Return to User)

ERETU returns from an event handler while changing the CPL to 3, establishing the **return context** that was in effect before FRED event delivery. The change of context includes updates to the segment registers CS, SS, or GS. Execution of ERETU causes an invalid-opcode exception (#UD) if FRED transitions are not enabled or if CPL > 0, and a general-protection exception (#GP) if CSL > 0.

1. ERETS will complete with indirect branch tracking enabled and not suppressed if CR4.CET = 1, IA32_S_CET.ENDBR_EN = 1, and IA32_S_CET.SUPPRESS = 0. If ERETS puts the indirect branch tracker in the WAIT_FOR_ENDBRANCH state, it will result in IA32_S_CET.TRACKER = 1.
2. If ERETS begins execution with RFLAGS.TF = 1, there will always be a single-step debug exception pending after ERETS, regardless of the values on the stack for RFLAGS.TF and CS.

ERETU begins by reading and checking the return state from the stack (Section 8.4.2.1). If supervisor shadow stacks are enabled, it then checks the validity of the old and new shadow-stack pointers (Section 8.4.2.2). Finally, ERETU establishes the return state by loading the appropriate registers (Section 8.4.2.3).

See “ERETU—Event Return to User” in Chapter 3 of Volume 2 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual for the detailed flow of the operation of ERETU.

8.4.2.1 Loading and Checking the Return State

ERETU operates on the stack frame created by FRED event delivery (Section 8.3.2.1), principally to restore the state saved in that stack frame (the return state).

When FRED event delivery completes, the return state is located at offset 8 from the RSP value. (RSP references an 8-byte field containing the event’s error code.) For this reason, ERETU begins by adding 8 to RSP so that it can access the return state directly.

After adding 8 to RSP, ERETU first pops from the regular stack (referenced by RSP) the return state that was saved by FRED event delivery. The state is checked and held by the processor to update register state when the instruction completes. The following items detail the state fields that are popped, from top (lowest address) to bottom (highest address), specifying the checks that are performed:

- The RIP of the return state (64 bits).
This field is checked once the new CS configuration is determined (see below).
- The augmented CS of the return state (64 bits).
Bits 15:0 contain the new CS selector itself. ERETU ensures that the value of bits 1:0 is 3; otherwise, a #GP occurs.
Bits 63:16 of this field must be zero; otherwise, a #GP occurs.
- The RFLAGS of the return state (64 bits).
Bit 1 of this field must be 1; bit 3, bit 5, bits 13:12 (IOPL), bit 15, bit 17 (VM), and bits 63:22 of the field must be zero; otherwise, a #GP occurs.

NOTE

Checking IOPL ensures that, when FRED transitions are enabled, IOPL is always 0 when CPL = 3.

- The RSP of the return state (64 bits). This field is not checked.
- The augmented SS of the return state (64 bits).
Bits 15:0 contain the new SS selector itself. ERETU ensures that the value of bits 1:0 is 3; otherwise, a #GP occurs.
Bit 17 and bit 18 of this field determine how ERETU manages certain events. See Section 8.4.2.3.
Bits 31:19 of this field must be zero; otherwise, a #GP occurs. ERETU ignores bit 16 of this field, as well as bits 63:32 (the event information).

After popping these fields, ERETU determines the configuration of the CS and SS segment registers for the return state according to one of these three cases:

1. If the selector popped for CS is `IA32_STAR[63:48] + 16` and the selector popped for SS is `IA32_STAR[63:48] + 8`, ERETU establishes CS and SS in a standard configuration for CPL 3 in 64-bit mode:
 - CS:
 - The selector is set to the selector popped for CS.
 - The base address is set to 0. The limit is set to `FFFFFH` and the G bit is set to 1.
 - The type is set to 11 (execute/read accessed code) and the S bit is set to 1.
 - The DPL is set to 3, the P and L bits are each set to 1, and the D bit is set to 0.
 - CS is made usable regardless of the value of the selector.
 - SS:

- The selector is set to the selector popped for SS.
 - The base address is set to 0. The limit is set to FFFFFFFH and the G bit is set to 1.
 - The type is set to 3 (read/write accessed data) and the S bit is set to 1.
 - The DPL is set to 3, and the P and B bits are each set to 1.
 - SS is made usable regardless of the value of the selector.
2. If the selector popped for CS is IA32_STAR[63:48] and the selector popped for SS is IA32_STAR[63:48] + 8, ERETU will establish CS and SS in a standard configuration for CPL s3 in compatibility mode:
- CS:
 - The selector is set to the selector popped for CS.
 - The base address is set to 0. The limit is set to FFFFFFFH and the G bit is set to 1.
 - The type is set to 11 (execute/read accessed code) and the S bit is set to 1.
 - The DPL is set to 3, the P and D bits are set to 1, and the L bit is set to 0.
 - CS is made usable regardless of the value of the selector.
 - SS:
 - The selector is set to the selector popped for SS.
 - The base address is set to 0. The limit is set to FFFFFFFH and the G bit is set to 1.
 - The type is set to 3 (read/write accessed data) and the S bit is set to 1.
 - The DPL is set to 3, and the P and B bits are each set to 1.
 - SS is made usable regardless of the value of the selector.
3. Otherwise, the selectors popped for CS and SS are used to load descriptors from the GDT or an LDT as would be done by an execution of IRET. These descriptor loads may cause ERETU to fault as would occur with IRET.

The RIP of the return state is then checked, depending on the mode to which ERETU is returning:

- If ERETU is returning to 64-bit mode (either case #1 above, or case #3, where the descriptor loaded for CS sets the L bit), a #GP with a zero error code occurs if the return RIP is not paging canonical.
- If ERETU is returning to compatibility mode (either case #2 above, or case #3, where the descriptor loaded for CS clears the L bit), the upper 32 bits of the return RIP are cleared. Following this, a #GP with a zero error code occurs if the truncated RIP value exceeds the new CS segment limit (taking into account that segment's G bit). (Such a #GP will never occur for case #2, as it establishes a limit of FFFFFFFFH.)

8.4.2.2 Checking the Shadow-Stack Pointers

If user shadow stacks are enabled, the SSP of the return state is the value of the IA32_PL3_SSP MSR. If the return is to compatibility mode, a #GP occurs if IA32_PL3_SSP[63:32] are not all zero.

If supervisor shadow stacks are enabled, ERETU compares the current (supervisor) SSP value and the value of the IA32_FRED_SSP0 MSR. If the values are not equal, a #CP occurs.

NOTE

Execution of ERETU does not access any shadow stack.

8.4.2.3 Establishing the Return State

After the stack operations described earlier, ERETU loads RIP, RFLAGS, RSP, CS, and SS with the values identified in Section 8.4.2.1. If the return is to compatibility mode, the upper 32 bits of each of RIP and RSP are cleared to zero. If user shadow stacks are enabled, SSP is loaded with the value for the return state as identified in Section 8.4.2.2 (from the IA32_PL3_SSP MSR).

ERETU swaps the value of the GS base address and that of the IA32_KERNEL_GS_BASE MSR.

Additional steps are performed based on the value of the popped SS field:

- If bit 17 of the field is 1 and ERETU would result in RFLAGS.TF = 1, a single-step trap will be pending upon completion of ERETU.¹
- If bit 18 of the field (above the selector) is 1, ERETU unblocks NMIs.

NOTE

Unlike IRET, ERETU does not make any of DS, ES, FS, or GS null if it is found to have DPL < 3. It is expected that a FRED-enabled operating system will return to CPL 3 (in compatibility mode) only when those segments all have DPL = 3.

8.4.2.4 Interactions Between ERETU and Other Instructions

ERETU provides the same instruction-ordering guarantees as SYSRET. Specifically, instructions following an execution of ERETU may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the ERETU have completed execution. (The later instructions may execute before data stored by the earlier instructions have become globally visible.)

Execution of the UMWAIT instruction will not wait (will not enter an implementation-dependent optimized state) if ERETU was executed before UMWAIT and after the most recent execution of the UMONITOR instruction.

8.5 FRED AND EXISTING INSTRUCTIONS

The Intel® 64 architecture defines numerous instructions that change the CPL. This section considers those instructions (and others) and describes how enabling FRED transitions affects either their operation or their usage. (Recall that FRED transitions are enabled if CR4.FRED = 1.)

- Section 8.5.1 identifies instructions that cannot be executed when FRED transitions are enabled.
- Section 8.5.2 considers far CALL, far JMP, far RET, and IRET. Enabling FRED transitions modifies the operation of these instructions. A FRED-enabled operating system cannot use them to change the CPL or to enter compatibility mode while CPL = 0.
- Section 8.5.3 discusses software interrupts and related instructions (INT n, INT3, INTO, and INT1). When FRED transitions are enabled, these instructions use FRED event delivery.
- Section 8.5.4 describes changes to the instructions SYSCALL and SYSENTER. When FRED transitions are enabled, these instructions also use FRED event delivery.
- Section 8.5.5 discusses interactions with the GETSEC instructions that enter or exit authenticated code execution mode.

8.5.1 Disallowed Instructions

When FRED transitions are enabled, execution of any of the following instructions causes an invalid-opcode exception (#UD): CLRSSBSY, SETSSBSY, SWAPGS, SYSEXIT, and SYSRET.

NOTE

CLRSSBSY, SETSSBSY, and SWAPGS account for anomalous situations that do not exist with FRED. In addition, CLRSSBSY and SETSSBSY operate on supervisor shadow-stack tokens, which are not used when FRED transitions are enabled.

8.5.2 Far CALL, IRET, Far JMP, and Far RET

Enabling FRED transitions disables the ability of certain existing instructions to change the CPL or to effect certain other mode changes.

1. If ERETU began execution with RFLAGS.TF = 1, there will always be a single-step debug exception pending after ERETU, regardless of the values on the stack for RFLAGS.TF and CS.

The far CALL instruction provides a mechanism by which user software can change to more privilege. The IRET and far RET instructions allow returns to the calling context, and this can change the CPL. The operation of these instructions (as well as that of far JMP) is modified when FRED transitions are enabled.

Far CALL can change CPL only if it references a call gate in the GDT or an LDT. (Far JMP can also reference a call gate, but an execution that does so cannot change the CPL.) When FRED transitions are enabled, any execution of far CALL or far JMP that references a call gate causes a general-protection exception (#GP). (The call-gate descriptor will not be modified.) The exception's error code will include the selector that referenced the call gate. This exception has the same priority as that of existing checks on the type of the target descriptor.

An execution of IRET or far RET changes the CPL if the RPL value of selector referencing the target code segment is greater than the CPL. When FRED transitions are enabled, such an execution causes #GP. (The code-segment descriptor will be accessed but not modified.) The exception's error code will include the selector that referenced the code segment. This exception has the same priority as that of the existing check that the RPL cannot be less than the CPL.

These restrictions imply that, when they are enabled, FRED transitions are the only ways to change the CPL and that it is impossible to change the CPL to 1 or to 2.

In addition, enabling FRED transitions disables the ability of software to enter compatibility mode while CPL = 0. An execution of far CALL, IRET, far JMP, or far RET causes a transition to compatibility mode if it loads CS from a code-segment descriptor in which the L bit is 0. (The L bit is bit 53 of the 64-bit descriptor.) When FRED transitions are enabled, such an execution with CPL = 0 causes #GP. The exception's error code will include the selector that referenced the code segment. This exception has the same priority as that of existing checks on the type of the target descriptor.

NOTE

The restrictions described in this section do not prevent use of far CALL, IRET, far JMP, or far RET to enter compatibility mode when the instruction is executed with CPL = 3 and does not reference a call gate. They can be used with CPL = 0 only if their execution does not change the CPL and stays in 64-bit mode.

8.5.3 Software Interrupts and Related Instructions

The Intel 64 architecture supports the following instructions that unprivileged software can use to invoke the operating system using the IDT:

- INT n (opcode CD followed by an immediate byte). There are 256 such software-interrupt instructions, one for each value n of the immediate byte (0–255).
- INT3 (opcode CC). This instruction generates a breakpoint exception (#BP) as a trap.
- INTO (opcode CE). If RFLAGS.OF = 1, this instruction generates an overflow exception (#OF) as a trap. If RFLAGS.OF = 0, this instruction does not generate an exception and control passes to the next instruction. Executing INTO in 64-bit mode causes an invalid-opcode exception (#UD), but the instruction can be used to generate #OF in compatibility mode.
- INT1 (opcode F1). This instruction generates a debug exception (#DB) as a trap. Hardware vendors may use INT1 for hardware debug.

When FRED transitions are enabled, an execution of any of these instructions (except for INTO if RFLAGS.OF = 0) results in FRED event delivery. Section 8.3.2.1 describes how the resulting FRED event delivery saves event information for these instructions (including the length of the instruction).

NOTE

IDT event delivery of INT n, INT3, and INTO checks the DPL field of the IDT gate and generates a general-protection exception (#GP) if it is less than the CPL. (Delivery of INT1 does not do this.) FRED event delivery does not use the IDT and thus does not perform this check. The event handlers of a FRED-enabled operating system should check the event type and vector to identify situations that would have resulted in a fault with IDT event delivery.

8.5.4 SYSCALL and SYSENTER

The Intel 64 architecture supports two instructions that unprivileged software can use to invoke the operating system without using the IDT: SYSCALL and SYSENTER.

When FRED transitions are enabled, the operation of SYSCALL is modified to use FRED event delivery (Section 8.3) in place of its existing operation. A FRED-enabled operating system should use ERETU (Section 8.4.2) instead of SYSRET to return after handling a system call invoked by an execution of SYSCALL with CPL = 3. (Recall that any execution of SYSRET causes #UD when FRED transitions are enabled. A FRED-enabled operating system will normally use ERETU to return from any event that occurred when CPL = 3.)

(Since the SYSRET instruction always changes the CPL to 3, use of SYSCALL has previously been effectively limited to CPL 3. When SYSCALL uses FRED event delivery, that instruction can be used effectively when CPL = 0. If this is done, a FRED-enabled operating system would naturally return with ERETS.)

The operation of SYSENTER is also modified to use FRED event delivery when FRED transitions are enabled. A FRED-enabled operating system should use ERETU instead of SYSEXIT to return after handling a system call invoked by an execution of SYSENTER with CPL = 3. (Recall that any execution of SYSEXIT causes #UD when FRED transitions are enabled.)

(Since the SYSEXIT instruction always changes the CPL to 3, use of SYSENTER has previously been effectively limited to CPL 3. When SYSENTER uses FRED event delivery, that instruction can be used effectively when CPL = 0. If this is done, a FRED-enabled operating system would naturally return with ERETS.)

8.5.5 GETSEC and Authenticated-Code Execution Mode

Software can invoke an authenticated-code module that operates in authenticated-code execution mode using the ENTERACCS and SENTER functions of the GETSEC instruction. The authenticated-code module leaves authenticated-code execution mode by executing the GETSEC function EXITAC.

After an authenticated-code module is invoked, events should not be delivered using FRED event delivery using a configuration that was established before entry to authenticated-code execution mode. Processors ensure that by disabling FRED transitions when entering authenticated-code execution mode. A subsequent execution of GETSEC[EXITAC] will re-enable FRED transitions if they had been enabled earlier.

To support this functionality, processors that support FRED transitions implement the features described in the remainder of this section.

A four-byte field called Cr4High is defined at offset 92 in the header of an authenticated-code module (ACM). This field is used by entry to and exit from authenticated-code execution mode.

Execution of either GETSEC[ENTERACCS] or GETSEC[SENDER] saves the current value of CR4[63:32] into the Cr4High field in the ACM header and then clears CR4.FRED to 0. This ensures that FRED event delivery will not be used in authenticated-code execution mode until the ACM has had the opportunity to configure and enable FRED.

If the logical processor entered authenticated-code execution mode using GETSEC[ENTERACCS], execution of GETSEC[EXITAC] reads the Cr4High field in the ACM header. If bit 0 of the Cr4High field in the ACM header (corresponding to CR4.FRED) is 1, a general-protection fault (#GP) occurs if GETSEC[EXITAC] is executed (and would thus end) outside IA-32e mode (IA32_EFER.LMA = 0). Otherwise, GETSEC[EXITAC] loads CR4.FRED with the value of that bit. (If the logical processor entered authenticated-code execution mode in some other manner, GETSEC[EXITAC] does not read the Cr4High field in the ACM header and does not modify CR4.)

8.6 LKGS: SUPPORT FOR MANAGING GS

64-bit operating systems and their applications use the GS segment for thread-local storage (TLS). Because the operating system and applications use the TLS at different addresses, they use different base addresses for that segment.

FRED transitions ensure that an operating system can always operate with its own GS base address:

- For events that occur while CPL = 3, FRED event delivery swaps the GS base address with the IA32_KERNEL_GS_BASE MSR.

- ERETU (the FRED transition that returns to CPL 3) also swaps the GS base address with the IA32_KERNEL_GS_BASE MSR.

An operating system can modify the GS base address of a user thread (e.g., as part of a context switch) by updating the IA32_KERNEL_GS_BASE MSR. However, the MOV GS instruction does not allow an operating system to modify other attributes of the GS segment without compromising its ability always to operate with its own GS base address. This is because that instruction updates the GS base address as well as those attributes.

The FRED architecture addresses this deficiency with the **LKGS** instruction (abbreviating “load into IA32_KERNEL_GS_BASE”). LKGS behaves like the MOV to GS instruction except that it loads the base address into the IA32_KERNEL_GS_BASE MSR instead of the GS segment’s descriptor cache.

Support for LKGS is enumerated with the feature flag CPUID.07H.01H:EAX.LKGS[18].

15. Updates to Chapter 9, Volume 3A

Change bars and text show changes to Chapter 9 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- In Section 9.5.1, "User-Interrupt Notification Identification", updated algorithm describing the response by the processor to a maskable interrupt by removing reference to IDT.

9.1 INTRODUCTION

This chapter provides details of an architectural feature called **user interrupts**.

This feature defines user interrupts as new events in the architecture. User interrupts are delivered to software operating in 64-bit mode with CPL = 3 without any change to segmentation state. An individual user interrupt is identified by a 6-bit user-interrupt vector, which is pushed on the stack as part of user-interrupt delivery. The UIRET (user-interrupt return) instruction reverses user-interrupt delivery.

System software configures the user-interrupt architecture with MSRs. An operating system (OS) may update the content of some of these MSRs when switching between OS-managed threads.

One of these MSRs references a data structure called the **user posted-interrupt descriptor (UPID)**. User interrupts for an OS-managed thread can be posted in the UPID associated with that thread. Such user interrupts will be delivered after receipt of an ordinary interrupt (identified in the UPID) called a **user-interrupt notification**.¹

System software can define operations to post user interrupts and to send user-interrupt notifications. In addition, the user-interrupt feature defines the SENDUIPI instruction, by which application software can send interprocessor user interrupts (user IPIs). An execution of SENDUIPI posts a user interrupt in a UPID and may send a user-interrupt notification.

(Platforms may include mechanisms to process external interrupts as either ordinary interrupts or user interrupts. Those processed as user interrupts would be posted in UPIDs and may result in user-interrupt notifications. Specifics of such mechanisms are outside of the scope of this manual.)

Section 9.2 explains how a processor enumerates support for user interrupts and how they are enabled by system software. Section 9.3 identifies the new processor state defined for user interrupts. Section 9.4 explains how a processor identifies and delivers user interrupts. Section 9.5 describes how a processor identifies and processes user-interrupt notifications. Section 9.6 enumerates new instructions that support management of user interrupts. Section 9.8 defines new support for user inter-processor interrupts (user IPIs).

9.2 ENUMERATION AND ENABLING

Software enables user interrupts by setting bit 25 (UINTR) in control register CR4. Setting CR4.UINTR enables user-interrupt delivery (Section 9.4.2), user-interrupt notification identification (Section 9.5.1), and the user-interrupt instructions (Section 9.6). It does not affect the accessibility of the user-interrupt MSRs (Section 9.3) by RDMSR, WRMSR or the XSAVE feature set.

Processor support for user interrupts is enumerated by CPUID.07H.00H:EDX[5]. If this bit is set, software can set CR4.UINTR to 1 and can access the user-interrupt MSRs using RDMSR and WRMSR (see Section 9.3).

The user-interrupt feature is XSAVE-managed (see Section 13.5). This implies that aspects of the feature are enumerated as part of enumeration of the XSAVE feature set. See Section 13.5.11 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for details.

9.3 USER-INTERRUPT STATE AND USER-INTERRUPT MSRS

The user-interrupt architecture defines the following new state. Some of this state can be accessed via the RDMSR and WRMSR instructions (through new user-interrupt MSRs detailed in Section 9.3.2) and some can be accessed using instructions described in Section 9.6.

1. For clarity, this chapter uses the term **ordinary interrupts** to refer to those events in the existing interrupt architecture, which are typically delivered to system software operating with CPL = 0.

9.3.1 User-Interrupt State

The following are the elements of the user-interrupt state (listed here independent of how they are accessed):

- **UIRR: user-interrupt request register.**

This value includes one bit for each of the 64 user-interrupt vectors. If $UIRR[i] = 1$, a user interrupt with vector i is requesting service. The notation **UIRRV** is used to refer to the position of the most significant bit set in UIRR; if $UIRR = 0$, $UIRRV = 0$.

- **UIF: user-interrupt flag.**

If $UIF = 0$, user-interrupt delivery is blocked; if $UIF = 1$, user interrupts may be delivered. User-interrupt delivery clears UIF, and the new UIRET instruction sets it. Section 9.6 defines other instructions for accessing UIF and Section 9.7 describes an enhancement that allows UIRET to maintain UIF as 0.

- **UIHANDLER: user-interrupt handler.**

This is the linear address of the user-interrupt handler. User-interrupt delivery loads this address into RIP.

- **UISTACKADJUST: user-interrupt stack adjustment.**

This value controls adjustment to the stack pointer (RSP) prior to user-interrupt delivery. It can be configured to load RSP with an alternate stack pointer or configured to prevent user-interrupt delivery from overwriting data above the current stack top.

The value UISTACKADJUST must be canonical. If bit 0 is 1, user-interrupt delivery loads RSP with UISTACKADJUST; otherwise, it subtracts UISTACKADJUST from RSP. Either way, user-interrupt delivery then aligns RSP to a 16-byte boundary. See Section 9.4.2 for details.

- **UINV: user-interrupt notification vector.**

This is the vector of the ordinary interrupts that are treated as user-interrupt notifications (Section 9.5.1). When the logical processor receives user-interrupt notification, it processes the user interrupts in the user posted-interrupt descriptor (UPID) referenced by UPIDADDR (see below and Section 9.5.2).

- **UPIDADDR: user posted-interrupt descriptor address.**

This is the linear address of the UPID that the logical processor consults upon receiving an ordinary interrupt with vector UINV.

- **UITTADDR: user-interrupt target table address.**

This is the linear address of user-interrupt target table (UITT), which the logical processor consults when software executes the SENDUIPI instruction (see Section 9.8).

- **UITTSZ: user-interrupt target table size.**

This value is the highest index of a valid entry in the UITT (see Section 9.8).

9.3.2 User-Interrupt MSRs

Some of the state elements identified in Section 9.3.1 can be accessed as user-interrupt MSRs using the RDMSR and WRMSR instructions:

- **IA32_UINTR_RR MSR (MSR address 985H).** This MSR is an interface to UIRR (64 bits).

Following a WRMSR to this MSR, the logical processor recognizes a pending user interrupt if and only if some bit is set in the MSR.

- **IA32_UINTR_HANDLER MSR (MSR address 986H).** This MSR is an interface to the UIHANDLER address. This is a linear address that must be canonical relative to the maximum linear-address width supported by the processor.¹ WRMSR to this MSR causes a general-protection fault (#GP) if its source operand does not meet this requirement.

- **IA32_UINTR_STACKADJUST MSR (MSR address 987H).** This MSR is an interface to the UISTACKADJUST value. This value includes a linear address that must be canonical relative to the maximum linear-address width supported by the processor. WRMSR to this MSR causes a #GP if its source operand does not meet this requirement.

1. CPUID.80000008H:EAX[15:8] enumerates the maximum linear-address width supported by the processor.

Bit 0 of this MSR corresponds to UISTACKADJUST[0], which controls how user-interrupt delivery updates the stack pointer. WRMSR may set it to either 0 or 1.

- IA32_UINTR_MISC MSR (MSR address 988H). This MSR is an interface to the UITSZ and UINV values. The MSR has the following format:

- Bits 31:0 are UITSZ.
- Bits 39:32 are UINV.
- Bits 63:40 are reserved. WRMSR causes a #GP if it would set any of those bits (if EDX[31:8] ≠ 000000H).

Because this MSR will share an 8-byte portion of the XSAVE area with UIF (see Section 13.5.11 of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1), bit 63 of the MSR will never be used and will always be reserved.

- IA32_UINTR_PD MSR (MSR address 989H). This MSR is an interface to the UPIDADDR address. This is a linear address that must be canonical relative to the maximum linear-address width supported by the processor. WRMSR to this MSR causes a #GP if its source operand does not meet this requirement.

Bits 5:0 of this MSR are reserved. WRMSR causes a #GP if it would set any of those bits (if EAX[5:0] ≠ 000000b).

- IA32_UINTR_TT MSR (MSR address 98AH). This MSR is an interface to the UITTADDR address (in addition, bit 0 enables SENDUIPI).

Bit 63:4 of this MSR holds the current value of UITTADDR. This is a linear address that must be canonical relative to the maximum linear-address width supported by the processor. WRMSR to this MSR causes a #GP if its source operand does not meet this requirement.

Bits 3:1 of this MSR are reserved. WRMSR causes a #GP if it would set any of those bits (if EAX[3:1] ≠ 000b).

Bit 0 of this MSR determines whether the SENDUIPI instruction is enabled. WRMSR may set it to either 0 or 1.

9.4 EVALUATION AND DELIVERY OF USER INTERRUPTS

A processor determines whether there is a user interrupt to deliver based on UIRR. Section 9.4.1 describes this recognition of pending user interrupts. Once a logical processor has recognized a pending user interrupt, it will deliver it on a subsequent instruction boundary by causing a control-flow change asynchronous to software execution. Section 9.4.2 details this process of user-interrupt delivery.

9.4.1 User-Interrupt Recognition

There is a user interrupt pending whenever UIRR ≠ 0.

Any instruction or operation that modifies UIRR updates the logical processor's recognition of a pending user interrupt. The following instructions and operations may do this:

- WRMSR to the IA32_UINTR_RR MSR (Section 9.3).
- XRSTORS of the user-interrupt state component.
- User-interrupt delivery (Section 9.4.2).
- User-interrupt notification processing (Section 9.5.2).
- VMX transitions that load the IA32_UINTR_RR MSR.

Each of these instructions or operations results in recognition of a pending user interrupt if it completes with UIRR ≠ 0; if it completes with UIRR = 0, no pending user interrupt is recognized.

Once recognized, a pending user interrupt may be delivered to software; see Section 9.4.2.

9.4.2 User-Interrupt Delivery

If CR4.UINTR = 1 and a user interrupt has been recognized (see Section 9.4.1), it will be delivered at an instruction boundary when the following conditions all hold: (1) UIF = 1; (2) there is no blocking by MOV SS or by POP SS¹;

(3) CPL = 3; (4) IA32_EFER.LMA = CS.L = 1 (the logical processor is in 64-bit mode); and (5) software is not executing inside an enclave.

User-interrupt delivery has priority just below that of ordinary interrupts. It wakes a logical processor from the states entered using the TPAUSE and UMWAIT instructions¹; it does not wake a logical processor in the shutdown state or in the wait-for-SIPI state.

User-interrupt delivery does not change CPL (it occurs entirely with CPL = 3). The following pseudocode details the behavior of user-interrupt delivery:

```

IF UIHANDLER is not canonical in current paging mode
    THEN #GP(0);
FI;
holdRSP := RSP;
IF UISTACKADJUST[0] = 1
    THEN RSP := UISTACKADJUST;
    ELSE RSP := RSP - UISTACKADJUST;
FI;
RSP := RSP & ~FH;           // force the stack to be 16-byte aligned
Push holdRSP;
Push RFLAGS;
Push RIP;
Push UIRR;                 // 64-bit push; upper 58 bits pushed as 0
IF shadow stack is enabled for CPL = 3
    THEN ShadowStackPush RIP;
FI;
IF end-branch is enabled for CPL = 3
    THEN IA32_U_CET.TRACKER := WAIT_FOR_ENDBRANCH;
FI;
UIRR[Vector] := 0;
IF UIRR = 0
    THEN cease recognition of any pending user interrupt;
FI;
UIF := 0;
RFLAGS.TF := 0;
RFLAGS.RF := 0;
RIP := UIHANDLER;

```

If UISTACKADJUST[0] = 0, user-interrupt delivery decrements RSP by UISTACKADJUST; otherwise, it loads RSP with UISTACKADJUST. In either case, user-interrupt delivery aligns RSP to a 16-byte boundary by clearing RSP[3:0].

User-interrupt delivery that occurs during transactional execution causes transactional execution to abort and a transition to a non-transactional execution. The transactional abort loads EAX as it would had it been due to an ordinary interrupt. User-interrupt delivery occurs after the transactional abort.

The stack accesses performed by user-interrupt delivery may incur faults (page faults, or stack faults due to canonicity violations). Before such a fault is delivered, RSP is restored to its original value (memory locations above the top of the stack may have been written). If such a fault produces an error code that uses the EXT bit, that bit will be cleared to 0.

-
1. Execution of the STI instruction does not block delivery of user interrupts for one instruction as it does ordinary interrupts. If a user interrupt is delivered immediately following execution of a STI instruction, ordinary interrupts are not blocked after delivery of the user interrupt.
 1. User-interrupt delivery occurs only if CPL = 3. Since the HLT and MWAIT instructions can be executed only if CPL = 0, a user interrupt can never be delivered when a logical processor is in an activity state that was entered using one of those instructions.

If a fault occurs during user-interrupt delivery, UIRR is not updated and UIF is not cleared and, as a result, the logical processor continues to recognize that a user interrupt is pending, and user-interrupt delivery will normally recur after the fault is handled.

If the shadow-stack feature of control-flow enforcement technology (CET) is enabled for CPL = 3, user-interrupt delivery pushes the return instruction pointer on the shadow stack. If indirect-branch-tracking feature of CET is enabled, user-interrupt delivery transitions the indirect branch tracker to the WAIT_FOR_ENDBRANCH state; an ENDBR64 instruction is expected as first instruction of the user-interrupt handler.

User-interrupt delivery can be tracked by Architectural Last Branch Records (LBRs), Intel® Processor Trace (Intel® PT), and Performance Monitoring. For both Intel PT and LBRs, user-interrupt delivery is recorded in precisely the same manner as ordinary interrupt delivery. Hence for LBRs, user interrupts fall into the OTHER_BRANCH category, which implies that IA32_LBR_CTL.OTHER_BRANCH[bit 22] must be set to record user-interrupt delivery, and that the IA32_LBR_x_INFO.BR_TYPE field will indicate OTHER_BRANCH for any recorded user interrupt. For Intel PT, control flow tracing must be enabled by setting IA32_RTIT_CTL.BranchEn[bit 13].

User-interrupt delivery will also increment performance counters for which counting BR_INST_RETIRED.FAR_BRANCH is enabled. Some implementations may have dedicated events for counting user-interrupt delivery; see processor-specific event lists at <https://github.com/intel/perfmon>.

9.5 USER-INTERRUPT NOTIFICATION IDENTIFICATION AND PROCESSING

User-interrupt posting is the process by which a platform agent (or software operating on a CPU) records user interrupts in a **user posted-interrupt descriptor** (UPID) in memory. The platform agent (or software) may send an ordinary interrupt (called a **user-interrupt notification**) to the logical processor on which the target of the user interrupt is operating.

Table 9-1 gives the format of a UPID.

Table 9-1. Format of User Posted-Interrupt Descriptor — UPID

Bit Position(s)	Name	Description
0	Outstanding notification	If this bit is set, there is a notification outstanding for one or more user interrupts in PIR.
1	Suppress notification	If this bit is set, agents (including SENDUIPI) should not send notifications when posting user interrupts in this descriptor.
15:2	Reserved	User-interrupt notification processing ignores these bits; must be zero for SENDUIPI.
23:16	Notification vector	Used by agents sending user-interrupt notifications (including SENDUIPI).
31:24	Reserved	User-interrupt notification processing ignores these bits; must be zero for SENDUIPI.
63:32	Notification destination	Target physical APIC ID – used by SENDUIPI. In xAPIC mode, bits 47:40 are the 8-bit APIC ID. In x2APIC mode, the entire field forms the 32-bit APIC ID.
127:64	Posted-interrupt requests (PIR)	One bit for each user-interrupt vector. There is a user-interrupt request for a vector if the corresponding bit is 1.

The notation **PIR** (posted-interrupt requests) refers to the 64 posted-interrupt requests in a UPID.

If an ordinary interrupt arrives while CR4.UINTR = IA32_EFER.LMA = 1, the logical processor determines whether the interrupt is a user-interrupt notification. This process is called **user-interrupt notification identification** and is described in Section 9.5.1.

Once a logical processor has identified a user-interrupt notification, it copies user interrupts in the UPID's PIR into UIRR. This process is called **user-interrupt notification processing** and is described in Section 9.5.2.

A logical processor is not interruptible during either user-interrupt notification identification or user-interrupt notification processing or between those operations (when they occur in succession).

9.5.1 User-Interrupt Notification Identification

If `CR4.UINTR = IA32_EFER.LMA = 1`, a logical processor performs user-interrupt notification identification when it receives an ordinary interrupt. The following algorithm describes the response by the processor to an ordinary maskable interrupt when `CR4.UINTR = IA32_EFER.LMA = 1`¹:

1. The local APIC is acknowledged; this provides the processor core with an interrupt vector, `V`.
2. If `V = UINV`, the logical processor continues to the next step. Otherwise, an interrupt with vector `V` is delivered normally; the remainder of this algorithm does not apply and user-interrupt notification processing does not occur.
3. The processor writes zero to the EOI register in the local APIC; this dismisses the interrupt with vector `V = UINV` from the local APIC.

User-interrupt notification identification involves acknowledgment of the local APIC and thus occurs only when ordinary interrupts are not masked.

If user-interrupt notification identification completes step #3, the logical processor then performs user-interrupt notification processing as described in Section 9.5.2.

An ordinary interrupt that occurs during transactional execution causes the transactional execution to abort and transition to a non-transactional execution. This occurs before user-interrupt notification identification.

An ordinary interrupt that occurs while software is executing inside an enclave causes an asynchronous enclave exit (AEX). This AEX occurs before user-interrupt notification identification.

9.5.2 User-Interrupt Notification Processing

Once a logical processor has identified a user-interrupt notification, it performs **user-interrupt notification processing** using the UPID at the linear address in the `IA32_UINTR_PD` MSR.

The following algorithm describes user-interrupt notification processing:

1. The logical processor clears the outstanding-notification bit (bit 0) in the UPID. This is done atomically so as to leave the remainder of the descriptor unmodified.
2. The logical processor reads PIR (bits 127:64 of the UPID) into a temporary register and writes all zeros to PIR. This is done atomically so as to ensure that each bit cleared in PIR is set in the temporary register.
3. If any bit is set in the temporary register, the logical processor sets in UIRR each bit corresponding to a bit set in the temporary register (e.g., with a logical OR) and recognizes a pending user interrupt (if it has not already done so).

The logical processor performs the steps above in an uninterruptible manner. Steps #1 and #2 may be combined into a single atomic step. If step #3 leads to recognition of a user interrupt, the processor may deliver that user interrupt on the following instruction boundary (see Section 9.4.2).

Although user-interrupt notification processing may occur at any privilege level, all of the memory accesses in steps #1 and #2 are performed with supervisor privilege.

Step #1 and step #2 each access the UPID using a linear address and may therefore incur faults (page faults, or general-protection faults due to canonicity violations). If such a fault produces an error code that uses the EXT bit, that bit will be set to 1.

If a fault occurs during user-interrupt notification processing, updates to architectural state performed by the earlier user-interrupt notification identification (Section 9.5.1) remain committed and are not undone; if such a fault occurs at step #2 (if it is not performed atomically with step #1), any update to architectural state performed by step #1 also remains committed. System software is advised to prevent such faults (e.g., by ensuring that no page fault occurs and that the linear address in the `IA32_UINTR_PD` MSR is canonical with respect to the paging mode in use).

1. If the interrupt arrives between iterations of a REP-prefixed string instruction, the processor first updates state as follows: RIP is loaded to reference the string instruction; RCX, RSI, and RDI are updated as appropriate to reflect the iterations completed; and RFLAGS.RF is set to 1.

If the user-interrupt notification identification that precedes user-interrupt notification processing occurred due to an ordinary interrupt that arrived while the logical processor was in the HLT state, the logical processor returns to the HLT state following user-interrupt notification processing.

9.6 USER-INTERRUPT INSTRUCTIONS

The user-interrupt feature defines instructions for control-flow transfer and access to new state. UIRET is an instruction to effect a return from a user-interrupt handler. CLUI, STUI, and TESTUI allow software to access UIF. SENDUIPI sends a user IPI. See Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D for details on the instructions' operation.

The following items provide high-level overviews of the instructions:

- UIRET pops from the stack the state saved by user-interrupt delivery (see Section 9.4.2) and loads those values into the corresponding registers (software should pop the user-interrupt vector from the stack before executing UIRET). Because RIP is one of those registers, UIRET effect a return to the that point from which the user interrupt was delivered.
- CLUI clears UIF.
- STUI sets UIF.
- TESTUI copies UIF to RFLAGS.CF.
- SENDUIPI is discussed in Section 9.8.

9.7 FLEXIBLE UPDATES OF UIF BY UIRET

There may be software usages that seek to return from the handling of a user interrupt while maintaining UIF as 0. This section describes an enhancement to UIRET that allows that.

This enhancement is supported if CPUID.07H.01H:EDX.UIRET_UIF[17] is enumerated as 1.

If the enhancement is supported, UIRET loads UIF with the value of the bit at position 1 in the RFLAGS image on the stack. (If the enhancement is not supported, UIRET ignores that bit in the RFLAGS image and always sets UIF to 1.)

The value of RFLAGS[1] is fixed as 1. All operations that save RFLAGS to memory save bit 1 as set; all operations that load RFLAGS leave bit 1 set.

This implies that user-interrupt delivery always saves on the stack an RFLAGS value that sets bit 1. If software does not modify this stack value, UIRET will set UIF to 1, even with the enhancement. Thus, the enhancement should not affect the operation of existing software, as long as that software does not modify the stack value saved for RFLAGS[1].

If a user-interrupt handler seek to return from the handling of a user interrupt while maintaining UIF as 0, it should modify the RFLAGS image on the stack to clear bit 1. Subsequent execution of UIRET will load UIF with 0, as long as the enhancement is supported.

Note that UIRET never modifies RFLAGS[1] (always leaving it with value 1) regardless of the stack value and regardless of whether enhancement is supported.

See Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D for details on the operation of the UIRET instruction.

9.8 USER IPIS

The SENDUIPI instruction sends a user interprocessor interrupt (IPI). The instruction uses a data structure called the user-interrupt target table (UITT). This table is located at the linear address UITTADDR and it comprises UITTSZ+1 16-byte entries (the values UITTADDR and UITTSZ are defined in Section 9.3.1). SENDUIPI uses the UITT entry (UITTE) indexed by the instruction's register operand. Each UITTE has the following format:

- Bit 0: V, a valid bit.

USER INTERRUPTS

- Bits 7:1 are reserved and must be 0.
- Bits 15:8: UV, the user-interrupt vector (in the range 0–63, so bits 15:14 must be 0).
- Bits 63:16 are reserved.
- Bits 127:64: UPIDADDR, the linear address of a UPID (64-byte aligned, so bits 69:64 must be 0).

SENDUIPI sends a user interrupt by posting a user interrupt with vector V in the UPID referenced by UPIDADDR and then sending, as an ordinary IPI, any notification interrupt specified in that UPID. Details appear in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B, 2C, & 2D.

16. Updates to Chapter 11, Volume 3A

Change bars and text show changes to Chapter 11 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- In Section 11.2.4, "Fast-String Operation and Out-of-Order Stores" and Section 11.2.4.1, "Memory-Ordering Model for String Operations on Write-Back (WB) Memory" updated content clarifying that fast-string operations may perform duplicate writes to certain memory addresses.

CHAPTER 11

MULTIPLE-PROCESSOR MANAGEMENT

The Intel 64 and IA-32 architectures provide mechanisms for managing and improving the performance of multiple processors connected to the same system bus. These include:

- Bus locking and/or cache coherency management for performing atomic operations on system memory.
- Serializing instructions.
- An advance programmable interrupt controller (APIC) located on the processor chip (see Chapter 13, “Advanced Programmable Interrupt Controller (APIC)”). This feature was introduced by the Pentium processor.
- A second-level cache (level 2, L2). For the Pentium 4, Intel Xeon, and P6 family processors, the L2 cache is included in the processor package and is tightly coupled to the processor. For the Pentium and Intel486 processors, pins are provided to support an external L2 cache.
- A third-level cache (level 3, L3). For Intel Xeon processors, the L3 cache is included in the processor package and is tightly coupled to the processor.
- Intel Hyper-Threading Technology. This extension to the Intel 64 and IA-32 architectures enables a single processor core to execute two or more threads concurrently (see Section 11.5, “Intel® Hyper-Threading Technology and Intel® Multi-Core Technology”).

These mechanisms are particularly useful in symmetric-multiprocessing (SMP) systems. However, they can also be used when an Intel 64 or IA-32 processor and a special-purpose processor (such as a communications, graphics, or video processor) share the system bus.

These multiprocessing mechanisms have the following characteristics:

- To maintain system memory coherency — When two or more processors are attempting simultaneously to access the same address in system memory, some communication mechanism or memory access protocol must be available to promote data coherency and, in some instances, to allow one processor to temporarily lock a memory location.
- To maintain cache consistency — When one processor accesses data cached on another processor, it must not receive incorrect data. If it modifies data, all other processors that access that data must receive the modified data.
- To allow predictable ordering of writes to memory — In some circumstances, it is important that memory writes be observed externally in precisely the same order as programmed.
- To distribute interrupt handling among a group of processors — When several processors are operating in a system in parallel, it is useful to have a centralized mechanism for receiving interrupts and distributing them to available processors for servicing.
- To increase system performance by exploiting the multi-threaded and multi-process nature of contemporary operating systems and applications.

The caching mechanism and cache consistency of Intel 64 and IA-32 processors are discussed in Chapter 14. The APIC architecture is described in Chapter 13. Bus and memory locking, serializing instructions, memory ordering, and Intel Hyper-Threading Technology are discussed in the following sections.

11.1 LOCKED ATOMIC OPERATIONS

The 32-bit IA-32 processors support locked atomic operations on locations in system memory. These operations are typically used to manage shared data structures (such as semaphores, segment descriptors, system segments, or page tables) in which two or more processors may try simultaneously to modify the same field or flag. The processor uses three interdependent mechanisms for carrying out locked atomic operations:

- Guaranteed atomic operations.
- Bus locking, using the LOCK# signal and the LOCK instruction prefix.

- Cache coherency protocols that ensure that atomic operations can be carried out on cached data structures (cache lock); this mechanism is present in the Pentium 4, Intel Xeon, and P6 family processors.

These mechanisms are interdependent in the following ways. Certain basic memory transactions (such as reading or writing a byte in system memory) are always guaranteed to be handled atomically. That is, once started, the processor guarantees that the operation will be completed before another processor or bus agent is allowed access to the memory location. The processor also supports bus locking for performing selected memory operations (such as a read-modify-write operation in a shared area of memory) that typically need to be handled atomically, but are not automatically handled this way. Because frequently used memory locations are often cached in a processor's L1 or L2 caches, atomic operations can often be carried out inside a processor's caches without asserting the bus lock. Here the processor's cache coherency protocols ensure that other processors that are caching the same memory locations are managed properly while atomic operations are performed on cached memory locations.

NOTE

Where there are contested lock accesses, software may need to implement algorithms that ensure fair access to resources in order to prevent lock starvation. The hardware provides no resource that guarantees fairness to participating agents. It is the responsibility of software to manage the fairness of semaphores and exclusive locking functions.

The mechanisms for handling locked atomic operations have evolved with the complexity of IA-32 processors. More recent IA-32 processors (such as the Pentium 4, Intel Xeon, and P6 family processors) and Intel 64 provide a more refined locking mechanism than earlier processors. These mechanisms are described in the following sections.

11.1.1 Guaranteed Atomic Operations

The Intel486 processor (and newer processors since) guarantees that the following basic memory operations will always be carried out atomically:

- Reading or writing a byte.
- Reading or writing a word aligned on a 16-bit boundary.
- Reading or writing a doubleword aligned on a 32-bit boundary.

The Pentium processor (and newer processors since) guarantees that the following additional memory operations will always be carried out atomically:

- Reading or writing a quadword aligned on a 64-bit boundary.
- 16-bit accesses to uncached memory locations that fit within a 32-bit data bus.

The P6 family processors (and newer processors since) guarantee that the following additional memory operation will always be carried out atomically:

- Unaligned 16-, 32-, and 64-bit accesses to cached memory that fit within a cache line.

Processors that enumerate support for Intel® AVX (by setting the feature flag CPUID.01H:ECX.AVX[28]) guarantee that the 16-byte memory operations performed by the following instructions will always be carried out atomically:

- MOVAPD, MOVAPS, and MOVDQA.
- VMOVAPD, VMOVAPS, and VMOVDQA when encoded with VEX.128.
- VMOVAPD, VMOVAPS, VMOVDQA32, and VMOVDQA64 when encoded with EVEX.128 and k0 (masking disabled).

(Note that these instructions require the linear addresses of their memory operands to be 16-byte aligned.)

Accesses to cacheable memory that are split across cache lines and page boundaries are not guaranteed to be atomic by the Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium M, Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors. The Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium M, Pentium 4, Intel Xeon, and P6 family processors provide bus control signals that permit external memory subsystems to make split accesses atomic; however, nonaligned data accesses will seriously impact the performance of the processor and should be avoided.

Except as noted above, an x87 instruction or an SSE instruction that accesses data larger than a quadword may be implemented using multiple memory accesses. If such an instruction stores to memory, some of the accesses may

complete (writing to memory) while another causes the operation to fault for architectural reasons (e.g., due an page-table entry that is marked “not present”). In this case, the effects of the completed accesses may be visible to software even though the overall instruction caused a fault. If TLB invalidation has been delayed (see Section 5.10.4.4), such page faults may occur even if all accesses are to the same page.

11.1.2 Bus Locking

Intel 64 and IA-32 processors provide a LOCK# signal that is asserted automatically during certain critical memory operations to lock the system bus or equivalent link. Assertion of this signal is called a **bus lock**. While this output signal is asserted, requests from other processors or bus agents for control of the bus are blocked. Software can specify other occasions when the LOCK semantics are to be followed by prepending the LOCK prefix to an instruction.

In the case of the Intel386, Intel486, and Pentium processors, explicitly locked instructions will result in the assertion of the LOCK# signal. It is the responsibility of the hardware designer to make the LOCK# signal available in system hardware to control memory accesses among processors.

For the P6 and more recent processor families, if the memory area being accessed is cached internally in the processor, the LOCK# signal is generally not asserted; instead, locking is only applied to the processor’s caches (see Section 11.1.4, “Effects of a LOCK Operation on Internal Processor Caches”). These processors will assert a bus lock for a locked access in either of the following situations: (1) the access is to multiple cache lines (a **split lock**); or (2) the access uses a memory type other than WB (a **UC lock**)¹.

11.1.2.1 Automatic Locking

The operations on which the processor automatically follows the LOCK semantics are as follows:

- When executing an XCHG instruction that references memory.
- When switching to a task, the processor tests and sets the busy flag in the type field of the TSS descriptor. To ensure that two processors do not switch to the same task simultaneously, the processor follows the LOCK semantics while testing and setting this flag.
- When loading a segment descriptor, the processor sets the accessed flag in the segment descriptor if the flag is clear. During this operation, the processor follows the LOCK semantics so that the descriptor will not be modified by another processor while it is being updated. For this action to be effective, operating-system procedures that update descriptors should use the following steps:
 - Use a locked operation to modify the access-rights byte to indicate that the segment descriptor is not-present, and specify a value for the type field that indicates that the descriptor is being updated.
 - Update the fields of the segment descriptor. (This operation may require several memory accesses; therefore, locked operations cannot be used.)
 - Use a locked operation to modify the access-rights byte to indicate that the segment descriptor is valid and present.
 - The Intel386 processor always updates the accessed flag in the segment descriptor, whether it is clear or not. The Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors only update this flag if it is not already set.
- The processor uses locked cycles to set the accessed and dirty flag in paging-structure entries.
- After an interrupt request, an interrupt controller may use the data bus to send the interrupt’s vector to the processor. The processor follows the LOCK semantics during this time to ensure that no other data appears on the data bus while the vector is being transmitted.

1. The term “UC lock” is used because the most common situation regards accesses to UC memory. Despite the name, locked accesses to WC, WP, and WT memory also cause bus locks.

11.1.2.2 Software Controlled Bus Locking

To explicitly force the LOCK semantics, software can use the LOCK prefix with the following instructions when they are used to modify a memory location. An invalid-opcode exception (#UD) is generated when the LOCK prefix is used with any other instruction or when no write operation is made to memory (that is, when the destination operand is in a register).

- The bit test and modify instructions (BTS, BTR, and BTC).
- The exchange instructions (XADD, CMPXCHG, CMPXCHG8B, and CMPXCHG16B).
- The LOCK prefix is automatically assumed for XCHG instruction.
- The following single-operand arithmetic and logical instructions: INC, DEC, NOT, and NEG.
- The following two-operand arithmetic and logical instructions: ADD, ADC, SUB, SBB, AND, OR, and XOR.

A locked instruction is guaranteed to lock only the area of memory defined by the destination operand, but may be interpreted by the system as a lock for a larger memory area.

Software should access semaphores (shared memory used for signaling between multiple processors) using identical addresses and operand lengths. For example, if one processor accesses a semaphore using a word access, other processors should not access the semaphore using a byte access.

NOTE

Do not implement semaphores using the WC memory type. Do not perform non-temporal stores to a cache line containing a location used to implement a semaphore.

The integrity of a bus lock is not affected by the alignment of the memory field. The LOCK semantics are followed for as many bus cycles as necessary to update the entire operand. However, it is recommended that locked accesses be aligned on their natural boundaries for better system performance:

- Any boundary for an 8-bit access (locked or otherwise).
- 16-bit boundary for locked word accesses.
- 32-bit boundary for locked doubleword accesses.
- 64-bit boundary for locked quadword accesses.

Locked operations are atomic with respect to all other memory operations and all externally visible events. Only instruction fetch and page table accesses can pass locked instructions. Locked instructions can be used to synchronize data written by one processor and read by another processor.

For the P6 family processors, locked operations serialize all outstanding load and store operations (that is, wait for them to complete). This rule is also true for the Pentium 4 and Intel Xeon processors, with one exception. Load operations that reference weakly ordered memory types (such as the WC memory type) may not be serialized.

Locked instructions should not be used to ensure that data written can be fetched as instructions.

NOTE

The locked instructions for the current versions of the Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors allow data written to be fetched as instructions. However, Intel recommends that developers who require the use of self-modifying code use a different synchronizing mechanism, described in the following sections.

11.1.2.3 Features to Disable Bus Locks

Because bus locks may adversely affect performance in certain situations, processors may support two features that system software can use to disable bus locking. These are called **split-lock disable** and **UC-lock disable**.

Support for split-lock disable is enumerated by IA32_CORE_CAPABILITIES[5].

Software enables split-lock disable by setting MSR_MEMORY_CTRL[29]. When this bit is set, a locked access to multiple cache lines causes a **bus-lock violation** that causes an alignment-check exception (#AC).¹ The locked access does not occur.

A processor enumerates support for UC-lock disable either by setting bit4 of the IA32_CORE_CAPABILITIES MSR (MSR index CFH) or by enumerating CPUID.07H.02H:EDX[6] as 1. The latter architectural form of enumeration (CPUID) is used beginning with processors based on Sierra Forest microarchitecture; earlier processors may use the older model-specific form (IA32_CORE_CAPABILITIES).

NOTE

No processor will both set IA32_CORE_CAPABILITIES[4] and enumerate CPUID.07H.02H:EDX[6] as 1.

If a processor enumerates support for UC-lock disable (in either way), software can enable UC-lock disable by setting MSR_MEMORY_CTRL[28]. When this bit is set, a locked access using a memory type other than WB causes a bus-lock violation, leading to a fault. The locked access does not occur. The specific fault that occurs depends on how UC-lock disable is enumerated:

- If IA32_CORE_CAPABILITIES[4] is read as 1, the UC lock results in a general-protection exception (#GP) with a zero error code.
- If CPUID.07H.02H:EDX[6] is enumerated as 1, the UC lock results in an #AC with an error code with value 4.¹

UC-lock disable does not apply to locked accesses to physical addresses specified in a VMCS. Such accesses include updates to accessed and dirty flags for EPT and those to posted-interrupt descriptors.

UC-lock disable is not enabled if CR0.CD = 1 or if MSR_PRMR_BASE_0[2:0] ≠ 6 (WB) when PRMRs are enabled. If either of those conditions hold, the processor ignores the value of MSR_MEMORY_CTRL[28].

Note that the #AC(0) due to split-lock disable or alignment check is higher priority than a #GP(0) or #AC(4) due to UC-lock disable. If both features are enabled, a locked access to multiple cache lines causes #AC(0) regardless of the memory type(s) being accessed.

While MSR_MEMORY_CTRL is not an architectural MSR, the behavior described above is consistent across processor models that enumerate the support in IA32_CORE_CAPABILITIES or CPUID.

In addition to these features that disable bus locks, there are features that allow software to detect when a bus lock has occurred. See Section 20.3.1.6 for information about OS bus-lock detection and Section 28.2 for information about the VMM bus-lock detection.

11.1.3 Handling Self- and Cross-Modifying Code

The act of a processor writing data into a currently executing code segment with the intent of executing that data as code is called **self-modifying code**. IA-32 processors exhibit model-specific behavior when executing self-modified code, depending upon how far ahead of the current execution pointer the code has been modified.

As processor microarchitectures become more complex and start to speculatively execute code ahead of the retirement point (as in P6 and more recent processor families), the rules regarding which code should execute, pre- or post-modification, become blurred. To write self-modifying code and ensure that it is compliant with current and future versions of the IA-32 architectures, use one of the following coding options:

(* OPTION 1 *)

Store modified code (as data) into code segment;
Jump to new code or an intermediate location;
Execute new code;

(* OPTION 2 *)

Store modified code (as data) into code segment;
Execute a serializing instruction; (* For example, CPUID instruction *)
Execute new code;

1. Other alignment-check exceptions occur only if CR0.AM = 1, EFLAGS.AC = 1, and CPL = 3. The alignment-check exceptions resulting from split-lock disable may occur even if CR0.AM = 0, EFLAGS.AC = 0, or CPL < 3. For these bus-lock violations, the error code is null, meaning that bits 31:2 of the error code are clear; bit 0 (EXT) may be set normally.

1. Bit 0 (EXT) may be set normally. The error code would have value 5 if the EXT bit is set.

The use of one of these options is not required for programs intended to run on the Pentium or Intel486 processors, but are recommended to ensure compatibility with the P6 and more recent processor families.

Self-modifying code will execute at a lower level of performance than non-self-modifying or normal code. The degree of the performance deterioration will depend upon the frequency of modification and specific characteristics of the code.

The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called **cross-modifying code**. As with self-modifying code, IA-32 processors exhibit model-specific behavior when executing cross-modifying code, depending upon how far ahead of the executing processors current execution pointer the code has been modified.

To write cross-modifying code and ensure that it is compliant with current and future versions of the IA-32 architecture, the following processor synchronization algorithm must be implemented:

```
(* Action of Modifying Processor *)
Memory_Flag := 0; (* Set Memory_Flag to value other than 1 *)
Store modified code (as data) into code segment;
Memory_Flag := 1;

(* Action of Executing Processor *)
WHILE (Memory_Flag ≠ 1)
    Wait for code to update;
ELIHW;
Execute serializing instruction; (* For example, CPUID instruction *)
Begin executing modified code;
```

(The use of this option is not required for programs intended to run on the Intel486 processor, but is recommended to ensure compatibility with the Pentium 4, Intel Xeon, P6 family, and Pentium processors.)

Like self-modifying code, cross-modifying code will execute at a lower level of performance than non-cross-modifying (normal) code, depending upon the frequency of modification and specific characteristics of the code.

The restrictions on self-modifying code and cross-modifying code also apply to the Intel 64 architecture.

11.1.4 Effects of a LOCK Operation on Internal Processor Caches

For the Intel486 and Pentium processors, the LOCK# signal is always asserted on the bus during a LOCK operation, even if the area of memory being locked is cached in the processor.

For the P6 and more recent processor families, if the area of memory being locked during a LOCK operation is cached in the processor that is performing the LOCK operation as write-back memory and is completely contained in a cache line, the processor may not assert the LOCK# signal on the bus. Instead, it will modify the memory location internally and allow its cache coherency mechanism to ensure that the operation is carried out atomically. This operation is called "cache locking." The cache coherency mechanism automatically prevents two or more processors that have cached the same area of memory from simultaneously modifying data in that area.

11.2 MEMORY ORDERING

The term **memory ordering** refers to the order in which the processor issues reads (loads) and writes (stores) through the system bus to system memory. The Intel 64 and IA-32 architectures support several memory-ordering models depending on the implementation of the architecture. For example, the Intel386 processor enforces **program ordering** (generally referred to as **strong ordering**), where reads and writes are issued on the system bus in the order they occur in the instruction stream under all circumstances.

To allow performance optimization of instruction execution, the IA-32 architecture allows departures from strong-ordering model called **processor ordering** in Pentium 4, Intel Xeon, and P6 family processors. These **processor-ordering** variations (called here the **memory-ordering model**) allow performance enhancing operations such as allowing reads to go ahead of buffered writes. The goal of any of these variations is to increase instruction execution speeds, while maintaining memory coherency, even in multiple-processor systems.

Section 11.2.1 and Section 11.2.2 describe the memory-ordering implemented by Intel486, Pentium, Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium 4, Intel Xeon, and P6 family processors. Section 11.2.3 gives examples illustrating the behavior of the memory-ordering model on IA-32 and Intel-64 processors. Section 11.2.4 considers the special treatment of stores for string operations and Section 11.2.5 discusses how memory-ordering behavior may be modified through the use of specific instructions.

11.2.1 Memory Ordering in the Intel® Pentium® and Intel486™ Processors

The Pentium and Intel486 processors follow the processor-ordered memory model; however, they operate as strongly-ordered processors under most circumstances. Reads and writes always appear in programmed order at the system bus—except for the following situation where processor ordering is exhibited. Read misses are permitted to go ahead of buffered writes on the system bus when all the buffered writes are cache hits and, therefore, are not directed to the same address being accessed by the read miss.

In the case of I/O operations, both reads and writes always appear in programmed order.

Software intended to operate correctly in processor-ordered processors (such as the Pentium 4, Intel Xeon, and P6 family processors) should not depend on the relatively strong ordering of the Pentium or Intel486 processors. Instead, it should ensure that accesses to shared variables that are intended to control concurrent execution among processors are explicitly required to obey program ordering through the use of appropriate locking or serializing operations (see Section 11.2.5, “Strengthening or Weakening the Memory-Ordering Model”).

11.2.2 Memory Ordering in P6 and More Recent Processor Families

The Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium 4, and P6 family processors also use a processor-ordered memory-ordering model that can be further defined as “write ordered with store-buffer forwarding.” This model can be characterized as follows.

In a single-processor system for memory regions defined as write-back cacheable, the memory-ordering model respects the following principles (**Note** the memory-ordering principles for single-processor and multiple-processor systems are written from the perspective of software executing on the processor, where the term “processor” refers to a logical processor. For example, a physical processor supporting multiple cores and/or Intel Hyper-Threading Technology is treated as a multi-processor systems.):

- Reads are not reordered with other reads.
- Writes are not reordered with older reads.
- Writes to memory are not reordered with other writes, with the following exceptions:
 - streaming stores (writes) executed with the non-temporal move instructions (MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, and MOVNTPD); and
 - string operations (see Section 11.2.4.1).
- No write to memory may be reordered with an execution of the CLFLUSH instruction; a write may be reordered with an execution of the CLFLUSHOPT instruction that flushes a cache line other than the one being written.¹ Executions of the CLFLUSH instruction are not reordered with each other. Executions of CLFLUSHOPT that access different cache lines may be reordered with each other. An execution of CLFLUSHOPT may be reordered with an execution of CLFLUSH that accesses a different cache line.
- Reads may be reordered with older writes to different locations but not with older writes to the same location.
- Reads or writes cannot be reordered with I/O instructions, locked instructions, or serializing instructions.
- Reads cannot pass earlier LFENCE and MFENCE instructions.
- Writes and executions of CLFLUSH and CLFLUSHOPT cannot pass earlier LFENCE, SFENCE, and MFENCE instructions.
- LFENCE instructions cannot pass earlier reads.
- SFENCE instructions cannot pass earlier writes or executions of CLFLUSH and CLFLUSHOPT.

1. Earlier versions of this manual specified that writes to memory may be reordered with executions of the CLFLUSH instruction. No processors implementing the CLFLUSH instruction allow such reordering.

- MFENCE instructions cannot pass earlier reads, writes, or executions of CLFLUSH and CLFLUSHOPT.

In a multiple-processor system, the following ordering principles apply:

- Individual processors use the same ordering principles as in a single-processor system.
- Writes by a single processor are observed in the same order by all processors.
- Writes from an individual processor are NOT ordered with respect to the writes from other processors.
- Memory ordering obeys causality (memory ordering respects transitive visibility).
- Any two stores are seen in a consistent order by processors other than those performing the stores
- Locked instructions have a total order.

See the example in Figure 11-1. Consider three processors in a system and each processor performs three writes, one to each of three defined locations (A, B, and C). Individually, the processors perform the writes in the same program order, but because of bus arbitration and other memory access mechanisms, the order that the three processors write the individual memory locations can differ each time the respective code sequences are executed on the processors. The final values in location A, B, and C would possibly vary on each execution of the write sequence.

The processor-ordering model described in this section is virtually identical to that used by the Pentium and Intel486 processors. The only enhancements in the Pentium 4, Intel Xeon, and P6 family processors are:

- Added support for speculative reads, while still adhering to the ordering principles above.
- Store-buffer forwarding, when a read passes a write to the same memory location.
- Out of order store from long string store and string move operations (see Section 11.2.4, “Fast-String Operation and Out-of-Order Stores,” below).

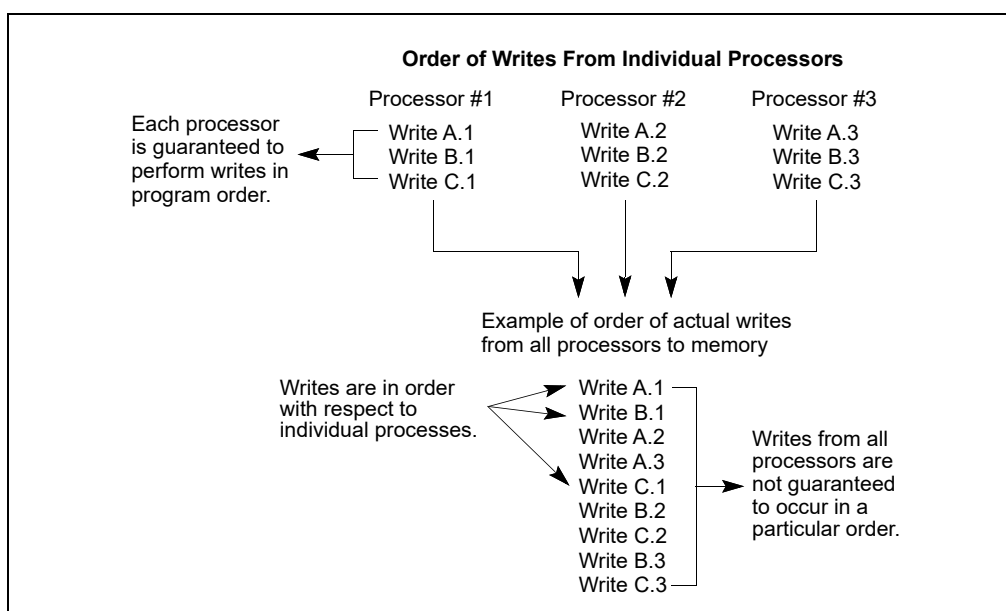


Figure 11-1. Example of Write Ordering in Multiple-Processor Systems

NOTE

In P6 processor family, store-buffer forwarding to reads of WC memory from streaming stores to the same address does not occur due to errata.

11.2.3 Examples Illustrating the Memory-Ordering Principles

This section provides a set of examples that illustrate the behavior of the memory-ordering principles introduced in Section 11.2.2. They are designed to give software writers an understanding of how memory ordering may affect the results of different sequences of instructions.

These examples are limited to accesses to memory regions defined as write-back cacheable (WB). (Section 11.2.3.1 describes other limitations on the generality of the examples.) The reader should understand that they describe only software-visible behavior. A logical processor may reorder two accesses even if one of examples indicates that they may not be reordered. Such an example states only that software cannot detect that such a reordering occurred. Similarly, a logical processor may execute a memory access more than once as long as the behavior visible to software is consistent with a single execution of the memory access.

11.2.3.1 Assumptions, Terminology, and Notation

As noted above, the examples in this section are limited to accesses to memory regions defined as write-back cacheable (WB). They apply only to ordinary loads stores and to locked read-modify-write instructions. They do not necessarily apply to any of the following: out-of-order stores for string instructions (see Section 11.2.4); accesses with a non-temporal hint; reads from memory by the processor as part of address translation (e.g., page walks); and updates to segmentation and paging structures by the processor (e.g., to update “accessed” bits).

The principles underlying the examples in this section apply to individual memory accesses and to locked read-modify-write instructions. The Intel-64 memory-ordering model guarantees that, for each of the following memory-access instructions, the constituent memory operation appears to execute as a single memory access:

- Instructions that read or write a single byte.
- Instructions that read or write a word (2 bytes) whose address is aligned on a 2 byte boundary.
- Instructions that read or write a doubleword (4 bytes) whose address is aligned on a 4 byte boundary.
- Instructions that read or write a quadword (8 bytes) whose address is aligned on an 8 byte boundary.

Any locked instruction (either the XCHG instruction or another read-modify-write instruction with a LOCK prefix) appears to execute as an indivisible and uninterruptible sequence of load(s) followed by store(s) regardless of alignment.

Other instructions may be implemented with multiple memory accesses. From a memory-ordering point of view, there are no guarantees regarding the relative order in which the constituent memory accesses are made. There is also no guarantee that the constituent operations of a store are executed in the same order as the constituent operations of a load.

Section 11.2.3.2 through Section 11.2.3.7 give examples using the MOV instruction. The principles that underlie these examples apply to load and store accesses in general and to other instructions that load from or store to memory. Section 11.2.3.8 and Section 11.2.3.9 give examples using the XCHG instruction. The principles that underlie these examples apply to other locked read-modify-write instructions.

This section uses the term “processor” to refer to a logical processor. The examples are written using Intel-64 assembly-language syntax and use the following notational conventions:

- Arguments beginning with an “r”, such as r1 or r2 refer to registers (e.g., EAX) visible only to the processor being considered.
- Memory locations are denoted with x, y, z.
- Stores are written as *mov [_x], val*, which implies that *val* is being stored into the memory location x.
- Loads are written as *mov r, [_x]*, which implies that the contents of the memory location x are being loaded into the register r.

As noted earlier, the examples refer only to software visible behavior. When the succeeding sections make statement such as “the two stores are reordered,” the implication is only that “the two stores appear to be reordered from the point of view of software.”

11.2.3.2 Neither Loads Nor Stores Are Reordered with Like Operations

The Intel-64 memory-ordering model allows neither loads nor stores to be reordered with the same kind of operation. That is, it ensures that loads are seen in program order and that stores are seen in program order. This is illustrated by the following example:

Example 11-1. Stores Are Not Reordered with Other Stores

Processor 0	Processor 1
mov [_x], 1	mov r1, [_y]
mov [_y], 1	mov r2, [_x]
Initially x = y = 0	
r1 = 1 and r2 = 0 is not allowed	

The disallowed return values could be exhibited only if processor 0’s two stores are reordered (with the two loads occurring between them) or if processor 1’s two loads are reordered (with the two stores occurring between them).

If r1 = 1, the store to y occurs before the load from y. Because the Intel-64 memory-ordering model does not allow stores to be reordered, the earlier store to x occurs before the load from y. Because the Intel-64 memory-ordering model does not allow loads to be reordered, the store to x also occurs before the later load from x. This r2 = 1.

11.2.3.3 Stores Are Not Reordered With Earlier Loads

The Intel-64 memory-ordering model ensures that a store by a processor may not occur before a previous load by the same processor. This is illustrated in Example 11-2.

Example 11-2. Stores Are Not Reordered with Older Loads

Processor 0	Processor 1
mov r1, [_x] mov [_y], 1	mov r2, [_y] mov [_x], 1
Initially x = y = 0 r1 = 1 and r2 = 1 is not allowed	

Assume r1 = 1.

- Because r1 = 1, processor 1's store to x occurs before processor 0's load from x.
- Because the Intel-64 memory-ordering model prevents each store from being reordered with the earlier load by the same processor, processor 1's load from y occurs before its store to x.
- Similarly, processor 0's load from x occurs before its store to y.
- Thus, processor 1's load from y occurs before processor 0's store to y, implying r2 = 0.

11.2.3.4 Loads May Be Reordered with Earlier Stores to Different Locations

The Intel-64 memory-ordering model allows a load to be reordered with an earlier store to a different location. However, loads are not reordered with stores to the same location.

The fact that a load may be reordered with an earlier store to a different location is illustrated by the following example:

Example 11-3. Loads May be Reordered with Older Stores

Processor 0	Processor 1
mov [_x], 1 mov r1, [_y]	mov [_y], 1 mov r2, [_x]
Initially x = y = 0 r1 = 0 and r2 = 0 is allowed	

At each processor, the load and the store are to different locations and hence may be reordered. Any interleaving of the operations is thus allowed. One such interleaving has the two loads occurring before the two stores. This would result in each load returning value 0.

The fact that a load may not be reordered with an earlier store to the same location is illustrated by the following example:

Example 11-4. Loads Are not Reordered with Older Stores to the Same Location

Processor 0
mov [_x], 1 mov r1, [_x]
Initially x = 0 r1 = 0 is not allowed

The Intel-64 memory-ordering model does not allow the load to be reordered with the earlier store because the accesses are to the same location. Therefore, r1 = 1 must hold.

11.2.3.5 Intra-Processor Forwarding Is Allowed

The memory-ordering model allows concurrent stores by two processors to be seen in different orders by those two processors; specifically, each processor may perceive its own store occurring before that of the other. This is illustrated by the following example:

Example 11-5. Intra-Processor Forwarding is Allowed

Processor 0	Processor 1
mov [_x], 1 mov r1, [_x] mov r2, [_y]	mov [_y], 1 mov r3, [_y] mov r4, [_x]
Initially x = y = 0 r2 = 0 and r4 = 0 is allowed	

The memory-ordering model imposes no constraints on the order in which the two stores appear to execute by the two processors. This fact allows processor 0 to see its store before seeing processor 1's, while processor 1 sees its store before seeing processor 0's. (Each processor is self consistent.) This allows r2 = 0 and r4 = 0.

In practice, the reordering in this example can arise as a result of store-buffer forwarding. While a store is temporarily held in a processor's store buffer, it can satisfy the processor's own loads but is not visible to (and cannot satisfy) loads by other processors.

11.2.3.6 Stores Are Transitively Visible

The memory-ordering model ensures transitive visibility of stores; stores that are causally related appear to all processors to occur in an order consistent with the causality relation. This is illustrated by the following example:

Example 11-6. Stores Are Transitively Visible

Processor 0	Processor 1	Processor 2
mov [_x], 1	mov r1, [_x] mov [_y], 1	mov r2, [_y] mov r3, [_x]
Initially x = y = 0 r1 = 1, r2 = 1, r3 = 0 is not allowed		

Assume that r1 = 1 and r2 = 1.

- Because r1 = 1, processor 0's store occurs before processor 1's load.
- Because the memory-ordering model prevents a store from being reordered with an earlier load (see Section 11.2.3.3), processor 1's load occurs before its store. Thus, processor 0's store causally precedes processor 1's store.
- Because processor 0's store causally precedes processor 1's store, the memory-ordering model ensures that processor 0's store appears to occur before processor 1's store from the point of view of all processors.
- Because r2 = 1, processor 1's store occurs before processor 2's load.
- Because the Intel-64 memory-ordering model prevents loads from being reordered (see Section 11.2.3.2), processor 2's load occur in order.
- The above items imply that processor 0's store to x occurs before processor 2's load from x. This implies that r3 = 1.

11.2.3.7 Stores Are Seen in a Consistent Order by Other Processors

As noted in Section 11.2.3.5, the memory-ordering model allows stores by two processors to be seen in different orders by those two processors. However, any two stores must appear to execute in the same order to all processors other than those performing the stores. This is illustrated by the following example:

Example 11-7. Stores Are Seen in a Consistent Order by Other Processors

Processor 0	Processor 1	Processor 2	Processor 3
mov [_x], 1	mov [_y], 1	mov r1, [_x] mov r2, [_y]	mov r3, [_y] mov r4, [_x]
Initially x = y = 0 r1 = 1, r2 = 0, r3 = 1, r4 = 0 is not allowed			

By the principles discussed in Section 11.2.3.2:

- Processor 2's first and second load cannot be reordered.
- Processor 3's first and second load cannot be reordered.
- If r1 = 1 and r2 = 0, processor 0's store appears to precede processor 1's store with respect to processor 2.
- Similarly, r3 = 1 and r4 = 0 imply that processor 1's store appears to precede processor 0's store with respect to processor 1.

Because the memory-ordering model ensures that any two stores appear to execute in the same order to all processors (other than those performing the stores), this set of return values is not allowed.

11.2.3.8 Locked Instructions Have a Total Order

The memory-ordering model ensures that all processors agree on a single execution order of all locked instructions, including those that are larger than 8 bytes or are not naturally aligned. This is illustrated by the following example:

Example 11-8. Locked Instructions Have a Total Order

Processor 0	Processor 1	Processor 2	Processor 3
xchg [_x], r1	xchg [_y], r2	mov r3, [_x] mov r4, [_y]	mov r5, [_y] mov r6, [_x]
Initially r1 = r2 = 1, x = y = 0 r3 = 1, r4 = 0, r5 = 1, r6 = 0 is not allowed			

Processor 2 and processor 3 must agree on the order of the two executions of XCHG. Without loss of generality, suppose that processor 0's XCHG occurs first.

- If r5 = 1, processor 1's XCHG into y occurs before processor 3's load from y.
- Because the Intel-64 memory-ordering model prevents loads from being reordered (see Section 11.2.3.2), processor 3's loads occur in order and, therefore, processor 1's XCHG occurs before processor 3's load from x.
- Since processor 0's XCHG into x occurs before processor 1's XCHG (by assumption), it occurs before processor 3's load from x. Thus, r6 = 1.

A similar argument (referring instead to processor 2's loads) applies if processor 1's XCHG occurs before processor 0's XCHG.

11.2.3.9 Loads and Stores Are Not Reordered with Locked Instructions

The memory-ordering model prevents loads and stores from being reordered with locked instructions that execute earlier or later. The examples in this section illustrate only cases in which a locked instruction is executed before a

load or a store. The reader should note that reordering is prevented also if the locked instruction is executed after a load or a store.

The first example illustrates that loads may not be reordered with earlier locked instructions:

Example 11-9. Loads Are not Reordered with Locks

Processor 0	Processor 1
xchg [_x], r1 mov r2, [_y]	xchg [_y], r3 mov r4, [_x]
Initially x = y = 0, r1 = r3 = 1 r2 = 0 and r4 = 0 is not allowed	

As explained in Section 11.2.3.8, there is a total order of the executions of locked instructions. Without loss of generality, suppose that processor 0’s XCHG occurs first.

Because the Intel-64 memory-ordering model prevents processor 1’s load from being reordered with its earlier XCHG, processor 0’s XCHG occurs before processor 1’s load. This implies r4 = 1.

A similar argument (referring instead to processor 2’s accesses) applies if processor 1’s XCHG occurs before processor 0’s XCHG.

The second example illustrates that a store may not be reordered with an earlier locked instruction:

Example 11-10. Stores Are not Reordered with Locks

Processor 0	Processor 1
xchg [_x], r1 mov [_y], 1	mov r2, [_y] mov r3, [_x]
Initially x = y = 0, r1 = 1 r2 = 1 and r3 = 0 is not allowed	

Assume r2 = 1.

- Because r2 = 1, processor 0’s store to y occurs before processor 1’s load from y.
- Because the memory-ordering model prevents a store from being reordered with an earlier locked instruction, processor 0’s XCHG into x occurs before its store to y. Thus, processor 0’s XCHG into x occurs before processor 1’s load from y.
- Because the memory-ordering model prevents loads from being reordered (see Section 11.2.3.2), processor 1’s loads occur in order and, therefore, processor 1’s XCHG into x occurs before processor 1’s load from x. Thus, r3 = 1.

11.2.4 Fast-String Operation and Out-of-Order Stores

Section 7.3.9.3 of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, described an optimization of repeated string operations called **fast-string operation**.

As explained in that section, the stores produced by fast-string operation may appear to execute out of order or be repeated. Software dependent upon sequential store ordering should not use string operations for the entire data structure to be stored. Data and semaphores should be separated. Order-dependent code should write to a discrete semaphore variable after any string operations to allow correctly ordered data to be seen by all processors. Atomicity of load and store operations is guaranteed only for native data elements of the string with native data size, and only if they are included in a single cache line.

Section 11.2.4.1 and Section 11.2.4.2 provide further explain and examples.

11.2.4.1 Memory-Ordering Model for String Operations on Write-Back (WB) Memory

This section deals with the memory-ordering model for string operations on write-back (WB) memory for the Intel 64 architecture.

The memory-ordering model respects the follow principles:

1. Stores within a single string operation may be executed out of order **or be repeated in the case of a fast string operation**.
2. Stores from separate string operations (for example, stores from consecutive string operations) do not execute out of order. All the stores from an earlier string operation will complete before any store from a later string operation.
3. String operations are not reordered with other store operations.

Fast string operations (e.g., string operations initiated with the MOVS/STOS instructions and the REP prefix) may be interrupted by exceptions or interrupts. The interrupts are precise but may be delayed - for example, the interruptions may be taken at cache line boundaries, after every few iterations of the loop, or after operating on every few bytes. Different implementations may choose different options, or may even choose not to delay interrupt handling, so software should not rely on the delay. When the interrupt/trap handler is reached, the source/destination registers point to the next string element to be operated on, while the EIP stored in the stack points to the string instruction, and the ECX register has the value it held following the last successful iteration. The return from that trap/interrupt handler should cause the string instruction to be resumed from the point where it was interrupted.

The string operation memory-ordering principles, (item 2 and 3 above) should be interpreted by taking the incoherency of fast string operations into account. For example, if a fast string operation gets interrupted after k iterations, then stores performed by the interrupt handler will become visible after the fast string stores from iteration 0 to k , and before the fast string stores from the $(k+1)$ th iteration onward.

Stores within a single string operation may execute out of order (item 1 above) only if fast string operation is enabled. Fast string operations are enabled/disabled through the IA32_MISC_ENABLE model specific register.

11.2.4.2 Examples Illustrating Memory-Ordering Principles for String Operations

The following examples uses the same notation and convention as described in Section 11.2.3.1.

In Example 11-11, processor 0 does one round of (128 iterations) doubleword string store operation via rep:stosd, writing the value 1 (value in EAX) into a block of 512 bytes from location $\_x$ (kept in ES:EDI) in ascending order. Since each operation stores a doubleword (4 bytes), the operation is repeated 128 times (value in ECX). The block of memory initially contained 0. Processor 1 is reading two memory locations that are part of the memory block being updated by processor 0, i.e, reading locations in the range $\_x$ to $(\_x+511)$.

Example 11-11. Stores Within a String Operation May be Reordered

Processor 0	Processor 1
rep:stosd $[\_x]$	mov r1, $[\_z]$ mov r2, $[\_y]$
Initially on processor 0: EAX = 1, ECX=128, ES:EDI = $\_x$ Initially $[\_x]$ to $511[\_x]$ = 0, $\_x \leq \_y < \_z < \_x+512$ $r1 = 1$ and $r2 = 0$ is allowed	

It is possible for processor 1 to perceive that the repeated string stores in processor 0 are happening out of order. Assume that fast string operations are enabled on processor 0.

In Example 11-12, processor 0 does two separate rounds of rep stosd operation of 128 doubleword stores, writing the value 1 (value in EAX) into the first block of 512 bytes from location $\_x$ (kept in ES:EDI) in ascending order. It then writes 1 into a second block of memory from $(\_x+512)$ to $(\_x+1023)$. All of the memory locations initially contain 0. The block of memory initially contained 0. Processor 1 performs two load operations from the two blocks of memory.

Example 11-12. Stores Across String Operations Are not Reordered

Processor 0	Processor 1
rep:stosd [_x] mov ecx, \$128 rep:stosd 512[_x]	mov r1, [_z] mov r2, [_y]
Initially on processor 0: EAX = 1, ECX=128, ES:EDI = _x Initially [_x] to 1023[_x]= 0, _x <= _y < _x+512 < _z < _x+1024 r1 = 1 and r2 = 0 is not allowed	

It is not possible in the above example for processor 1 to perceive any of the stores from the later string operation (to the second 512 block) in processor 0 before seeing the stores from the earlier string operation to the first 512 block.

The above example assumes that writes to the second block (_x+512 to _x+1023) does not get executed while processor 0's string operation to the first block has been interrupted. If the string operation to the first block by processor 0 is interrupted, and a write to the second memory block is executed by the interrupt handler, then that change in the second memory block will be visible before the string operation to the first memory block resumes.

In Example 11-13, processor 0 does one round of (128 iterations) doubleword string store operation via rep:stosd, writing the value 1 (value in EAX) into a block of 512 bytes from location _x (kept in ES:EDI) in ascending order. It then writes to a second memory location outside the memory block of the previous string operation. Processor 1 performs two read operations, the first read is from an address outside the 512-byte block but to be updated by processor 0, the second ready is from inside the block of memory of string operation.

Example 11-13. String Operations Are not Reordered with later Stores

Processor 0	Processor 1
rep:stosd [_x] mov [_z], \$1	mov r1, [_z] mov r2, [_y]
Initially on processor 0: EAX = 1, ECX=128, ES:EDI = _x Initially [_y] = [_z] = 0, [_x] to 511[_x]= 0, _x <= _y < _x+512, _z is a separate memory location r1 = 1 and r2 = 0 is not allowed	

Processor 1 cannot perceive the later store by processor 0 until it sees all the stores from the string operation. Example 11-13 assumes that processor 0's store to [_z] is not executed while the string operation has been interrupted. If the string operation is interrupted and the store to [_z] by processor 0 is executed by the interrupt handler, then changes to [_z] will become visible before the string operation resumes.

Example 11-14 illustrates the visibility principle when a string operation is interrupted.

Example 11-14. Interrupted String Operation

Processor 0	Processor 1
rep:stosd [_x] // interrupted before es:edi reach _y mov [_z], \$1 // interrupt handler	mov r1, [_z] mov r2, [_y]
Initially on processor 0: EAX = 1, ECX=128, ES:EDI = _x Initially [_y] = [_z] = 0, [_x] to 511[_x]= 0, _x <= _y < _x+512, _z is a separate memory location r1 = 1 and r2 = 0 is allowed	

In Example 11-14, processor 0 started a string operation to write to a memory block of 512 bytes starting at address `_x`. Processor 0 got interrupted after `k` iterations of store operations. The address `_y` has not yet been updated by processor 0 when processor 0 got interrupted. The interrupt handler that took control on processor 0 writes to the address `_z`. Processor 1 may see the store to `_z` from the interrupt handler, before seeing the remaining stores to the 512-byte memory block that are executed when the string operation resumes.

Example 11-15 illustrates the ordering of string operations with earlier stores. No store from a string operation can be visible before all prior stores are visible.

Example 11-15. String Operations Are not Reordered with Earlier Stores

Processor 0	Processor 1
<code>mov [_z], \$1</code> <code>rep:stosd [_x]</code>	<code>mov r1, [_y]</code> <code>mov r2, [_z]</code>
Initially on processor 0: <code>EAX = 1, ECX=128, ES:EDI = _x</code> Initially <code>[_y] = [_z] = 0, [_x] to 511[_x] = 0, _x <= _y < _x+512, _z</code> is a separate memory location <code>r1 = 1 and r2 = 0</code> is not allowed	

11.2.5 Strengthening or Weakening the Memory-Ordering Model

The Intel 64 and IA-32 architectures provide several mechanisms for strengthening or weakening the memory-ordering model to handle special programming situations. These mechanisms include:

- The I/O instructions, locked instructions, the LOCK prefix, and serializing instructions force stronger ordering on the processor.
- The SFENCE instruction (introduced to the IA-32 architecture in the Pentium III processor) and the LFENCE and MFENCE instructions (introduced in the Pentium 4 processor) provide memory-ordering and serialization capabilities for specific types of memory operations.
- The memory type range registers (MTRRs) can be used to strengthen or weaken memory ordering for specific area of physical memory (see Section 14.11, “Memory Type Range Registers (MTRRs)”). MTRRs are available only in the Pentium 4, Intel Xeon, and P6 family processors.
- The page attribute table (PAT) can be used to strengthen memory ordering for a specific page or group of pages (see Section 14.12, “Page Attribute Table (PAT)”). The PAT is available only in the Pentium 4, Intel Xeon, and Pentium III processors.

These mechanisms can be used as follows:

Memory mapped devices and other I/O devices on the bus are often sensitive to the order of writes to their I/O buffers. I/O instructions can be used to (the IN and OUT instructions) impose strong write ordering on such accesses as follows. Prior to executing an I/O instruction, the processor waits for all previous instructions in the program to complete and for all buffered writes to drain to memory. Only instruction fetch and page tables walks can pass I/O instructions. Execution of subsequent instructions do not begin until the processor determines that the I/O instruction has been completed.

Synchronization mechanisms in multiple-processor systems may depend upon a strong memory-ordering model. Here, a program can use a locked instruction such as the XCHG instruction or the LOCK prefix to ensure that a read-modify-write operation on memory is carried out atomically. Locked instructions typically operate like I/O instructions in that they wait for all previous memory accesses to complete and for all buffered writes to drain to memory (see Section 11.1.2, “Bus Locking”). Unlike I/O operations, locked instructions do not wait for all previous instructions to complete execution.

Program synchronization can also be carried out with serializing instructions (see Section 11.3). These instructions are typically used at critical procedure or task boundaries to force completion of all previous instructions before a jump to a new section of code or a context switch occurs. Like the I/O instructions, the processor waits until all previous instructions have been completed and all buffered writes have been drained to memory before executing the serializing instruction.

The SFENCE, LFENCE, and MFENCE instructions provide a performance-efficient way of ensuring load and store memory ordering between routines that produce weakly-ordered results and routines that consume that data. The functions of these instructions are as follows:

- **SFENCE** — Serializes all store (write) operations that occurred prior to the SFENCE instruction in the program instruction stream, but does not affect load operations.
- **LFENCE** — Serializes all load (read) operations that occurred prior to the LFENCE instruction in the program instruction stream, but does not affect store operations.¹
- **MFENCE** — Serializes all store and load operations that occurred prior to the MFENCE instruction in the program instruction stream.

Note that the SFENCE, LFENCE, and MFENCE instructions provide a more efficient method of controlling memory ordering than the CPUID instruction.

The MTRRs were introduced in the P6 family processors to define the cache characteristics for specified areas of physical memory. The following are two examples of how memory types set up with MTRRs can be used strengthen or weaken memory ordering for the Pentium 4, Intel Xeon, and P6 family processors:

- The strong uncached (UC) memory type forces a strong-ordering model on memory accesses. Here, all reads and writes to the UC memory region appear on the bus and out-of-order or speculative accesses are not performed. This memory type can be applied to an address range dedicated to memory mapped I/O devices to force strong memory ordering.
- For areas of memory where weak ordering is acceptable, the write back (WB) memory type can be chosen. Here, reads can be performed speculatively and writes can be buffered and combined. For this type of memory, cache locking is performed on atomic (locked) operations that do not split across cache lines, which helps to reduce the performance penalty associated with the use of the typical synchronization instructions, such as XCHG, that lock the bus during the entire read-modify-write operation. With the WB memory type, the XCHG instruction locks the cache instead of the bus if the memory access is contained within a cache line.

The PAT was introduced in the Pentium III processor to enhance the caching characteristics that can be assigned to pages or groups of pages. The PAT mechanism typically used to strengthen caching characteristics at the page level with respect to the caching characteristics established by the MTRRs. Table 14-7 shows the interaction of the PAT with the MTRRs.

Intel recommends that software written to run on Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium 4, Intel Xeon, and P6 family processors assume the processor-ordering model or a weaker memory-ordering model. The Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium 4, Intel Xeon, and P6 family processors do not implement a strong memory-ordering model, except when using the UC memory type. Despite the fact that Pentium 4, Intel Xeon, and P6 family processors support processor ordering, Intel does not guarantee that future processors will support this model. To make software portable to future processors, it is recommended that operating systems provide critical region and resource control constructs and API's (application program interfaces) based on I/O, locking, and/or serializing instructions be used to synchronize access to shared areas of memory in multiple-processor systems. Also, software should not depend on processor ordering in situations where the system hardware does not support this memory-ordering model.

11.3 SERIALIZING INSTRUCTIONS

The Intel 64 and IA-32 architectures define several **serializing instructions**. These instructions force the processor to complete all modifications to flags, registers, and memory by previous instructions and to drain all buffered writes to memory before the next instruction is fetched and executed. For example, when a MOV to control register instruction is used to load a new value into control register CR0 to enable protected mode, the processor must perform a serializing operation before it enters protected mode. This serializing operation ensures

1. Specifically, LFENCE does not execute until all prior instructions have completed locally, and no later instruction begins execution until LFENCE completes. As a result, an instruction that loads from memory and that precedes an LFENCE receives data from memory prior to completion of the LFENCE. An LFENCE that follows an instruction that stores to memory might complete before the data being stored have become globally visible. Instructions following an LFENCE may be fetched from memory before the LFENCE, but they will not execute until the LFENCE completes.

that all operations that were started while the processor was in real-address mode are completed before the switch to protected mode is made.

The concept of serializing instructions was introduced into the IA-32 architecture with the Pentium processor to support parallel instruction execution. Serializing instructions have no meaning for the Intel486 and earlier processors that do not implement parallel instruction execution.

It is important to note that executing of serializing instructions on P6 and more recent processor families constrain speculative execution because the results of speculatively executed instructions are discarded. The following instructions are serializing instructions:

- **Privileged serializing instructions** — INVD, INVEPT, INVLPG, INVVPID, LGDT, LIDT, LLDT, LTR, MOV (to control register, with the exception of MOV CR8¹), MOV (to debug register), WBINVD, and WRMSR².
- **Non-privileged serializing instructions** — CPUID, IRET, RSM, and SERIALIZE.

When the processor serializes instruction execution, it ensures that all pending memory transactions are completed (including writes stored in its store buffer) before it executes the next instruction. Nothing can pass a serializing instruction and a serializing instruction cannot pass any other instruction (read, write, instruction fetch, or I/O). For example, CPUID can be executed at any privilege level to serialize instruction execution with no effect on program flow, except that the EAX, EBX, ECX, and EDX registers are modified.

The following instructions are memory-ordering instructions, not serializing instructions. These drain the data memory subsystem. They do not serialize the instruction execution stream:³

- **Non-privileged memory-ordering instructions** — SFENCE, LFENCE, and MFENCE.

The SFENCE, LFENCE, and MFENCE instructions provide more granularity in controlling the serialization of memory loads and stores (see Section 11.2.5, “Strengthening or Weakening the Memory-Ordering Model”).

The following additional information is worth noting regarding serializing instructions:

- The processor does not write back the contents of modified data in its data cache to external memory when it serializes instruction execution. Software can force modified data to be written back by executing the WBINVD instruction, which is a serializing instruction. The amount of time or cycles for WBINVD to complete will vary due to the size of different cache hierarchies and other factors. As a consequence, the use of the WBINVD instruction can have an impact on interrupt/event response time.
- When an instruction is executed that enables or disables paging (that is, changes the PG flag in control register CR0), the instruction should be followed by a jump instruction. The target instruction of the jump instruction is fetched with the new setting of the PG flag (that is, paging is enabled or disabled), but the jump instruction itself is fetched with the previous setting. The Pentium 4, Intel Xeon, and P6 family processors do not require the jump operation following the move to register CR0 (because any use of the MOV instruction in a Pentium 4, Intel Xeon, or P6 family processor to write to CR0 is completely serializing). However, to maintain backwards and forward compatibility with code written to run on other IA-32 processors, it is recommended that the jump operation be performed.
- Whenever an instruction is executed to change the contents of CR3 while paging is enabled, the next instruction is fetched using the translation tables that correspond to the new value of CR3. Therefore the next instruction and the sequentially following instructions should have a mapping based upon the new value of CR3. (Global entries in the TLBs are not invalidated, see Section 5.10.4, “Invalidation of TLBs and Paging-Structure Caches.”)
- The Pentium processor and more recent processor families use branch-prediction techniques to improve performance by prefetching the destination of a branch instruction before the branch instruction is executed. Consequently, instruction execution is not deterministically serialized when a branch instruction is executed.

1. MOV CR8 is not defined architecturally as a serializing instruction.

2. An execution of WRMSR to any non-serializing MSR is not serializing. Non-serializing MSRs include the following: IA32_SPEC_CTRL MSR (MSR index 48H), IA32_PRED_CMD MSR (MSR index 49H), IA32_TSX_CTRL MSR (MSR index 122H), IA32_TSC_DEADLINE MSR (MSR index 6E0H), IA32_PKRS MSR (MSR index 6E1H), IA32_HWP_REQUEST MSR (MSR index 774H), or any of the x2APIC MSRs (MSR indices 802H to 83FH).

3. LFENCE does provide some guarantees on instruction ordering. It does not execute until all prior instructions have completed locally, and no later instruction begins execution until LFENCE completes.

11.4 MULTIPLE-PROCESSOR (MP) INITIALIZATION

The IA-32 architecture (beginning with the P6 family processors) defines a multiple-processor (MP) initialization protocol called the Multiprocessor Specification Version 1.4. This specification defines the boot protocol to be used by IA-32 processors in multiple-processor systems. (Here, **multiple processors** is defined as two or more processors.) The MP initialization protocol has the following important features:

- It supports controlled booting of multiple processors without requiring dedicated system hardware.
- It allows hardware to initiate the booting of a system without the need for a dedicated signal or a predefined boot processor.
- It allows all IA-32 processors to be booted in the same manner, including those supporting Intel Hyper-Threading Technology.
- The MP initialization protocol also applies to MP systems using Intel 64 processors.

The mechanism for carrying out the MP initialization protocol differs depending on the Intel processor generations. The following bullets summarize the evolution of the changes:

- **For P6 family or older processors supporting MP operations**— The selection of the BSP and APs (see Section 11.4.1, “BSP and AP Processors”) is handled through arbitration on the APIC bus, using BIPI and FIPI messages. These processor generations have CPUID signatures of (family=06H, extended_model=0, model<=0DH), or family <06H. See Section 11.11.1, “Overview of the MP Initialization Process for P6 Family Processors,” for a complete discussion of MP initialization for P6 family processors.
- **Early generations of IA processors with family 0FH** — The selection of the BSP and APs (see Section 11.4.1, “BSP and AP Processors”) is handled through arbitration on the system bus, using BIPI and FIPI messages (see Section 11.4.3, “MP Initialization Protocol Algorithm for MP Systems”). These processor generations have CPUID signatures of family=0FH, model=0H, stepping<=09H.
- **Later generations of IA processors with family 0FH, and IA processors with system bus** — The selection of the BSP and APs is handled through a special system bus cycle, without using BIPI and FIPI message arbitration (see Section 11.4.3, “MP Initialization Protocol Algorithm for MP Systems”). These processor generations have CPUID signatures of family=0FH with (model=0H, stepping>=0AH) or (model >0, all steppings); or family=06H, extended_model=0, model>=0EH.
- **All other modern IA processor generations supporting MP operations**— The selection of the BSP and APs in the system is handled by platform-specific arrangement of the combination of hardware, BIOS, and/or configuration input options. The basis of the selection mechanism is similar to those of the Later generations of family 0FH and other Intel processor using system bus (see Section 11.4.3, “MP Initialization Protocol Algorithm for MP Systems”). These processor generations have CPUID signatures of family=06H, extended_model>0.

The family, model, and stepping ID for a processor is returned in the EAX register by CPUID.01H.

11.4.1 BSP and AP Processors

The MP initialization protocol defines two classes of processors: the bootstrap processor (BSP) and the application processors (APs). Following a power-up or RESET of an MP system, system hardware dynamically selects one of the processors on the system bus as the BSP. The remaining processors are designated as APs.

As part of the BSP selection mechanism, the BSP flag is set in the IA32_APIC_BASE MSR (see Figure 13-5) of the BSP, indicating that it is the BSP. This flag is cleared for all other processors.

The BSP executes the BIOS’s boot-strap code to configure the APIC environment, sets up system-wide data structures, and starts and initializes the APs. When the BSP and APs are initialized, the BSP then begins executing the operating-system initialization code.

Following a power-up or reset, the APs complete a minimal self-configuration, then wait for a startup signal (a SIPI message) from the BSP processor. Upon receiving a SIPI message, an AP executes the BIOS AP configuration code, which ends with the AP being placed in halt state.

For Intel 64 and IA-32 processors supporting Intel Hyper-Threading Technology, the MP initialization protocol treats each of the logical processors on the system bus or coherent link domain as a separate processor (with a unique

APIC ID). During boot-up, one of the logical processors is selected as the BSP and the remainder of the logical processors are designated as APs.

11.4.2 MP Initialization Protocol Requirements and Restrictions

The MP initialization protocol imposes the following requirements and restrictions on the system:

- The MP protocol is executed only after a power-up or RESET. If the MP protocol has completed and a BSP is chosen, subsequent INITs (either to a specific processor or system wide) do not cause the MP protocol to be repeated. Instead, each logical processor examines its BSP flag (in the IA32_APIC_BASE MSR) to determine whether it should execute the BIOS boot-strap code (if it is the BSP) or enter a wait-for-SIPI state (if it is an AP).
- All devices in the system that are capable of delivering interrupts to the processors must be inhibited from doing so for the duration of the MP initialization protocol. The time during which interrupts must be inhibited includes the window between when the BSP issues an INIT-SIPI-SIPI sequence to an AP and when the AP responds to the last SIPI in the sequence.

11.4.3 MP Initialization Protocol Algorithm for MP Systems

Following a power-up or RESET of an MP system, the processors in the system execute the MP initialization protocol algorithm to initialize each of the logical processors on the system bus or coherent link domain. In the course of executing this algorithm, the following boot-up and initialization operations are carried out:

1. Each logical processor is assigned a unique APIC ID, based on system topology. The unique ID is a 32-bit value if the processor supports CPUID.0BH, otherwise the unique ID is an 8-bit value. (see Section 11.4.5, "Identifying Logical Processors in an MP System").
2. Each logical processor is assigned a unique arbitration priority based on its APIC ID.
3. Each logical processor executes its internal BIST simultaneously with the other logical processors in the system.
4. Upon completion of the BIST, the logical processors use a hardware-defined selection mechanism to select the BSP and the APs from the available logical processors on the system bus. The BSP selection mechanism differs depending on the family, model, and stepping IDs of the processors, as follows:
 - Later generations of IA processors within family 0FH (see Section 11.4), IA processors with system bus (family=06H, extended_model=0, model>=0EH), or all other modern Intel processors (family=06H, extended_model>0):
 - The logical processors begin monitoring the BNR# signal, which is toggling. When the BNR# pin stops toggling, each processor attempts to issue a NOP special cycle on the system bus.
 - The logical processor with the highest arbitration priority succeeds in issuing a NOP special cycle and is nominated the BSP. This processor sets the BSP flag in its IA32_APIC_BASE MSR, then fetches and begins executing BIOS boot-strap code, beginning at the reset vector (physical address FFFF FFF0H).
 - The remaining logical processors (that failed in issuing a NOP special cycle) are designated as APs. They leave their BSP flags in the clear state and enter a "wait-for-SIPI state."
 - Early generations of IA processors within family 0FH (family=0FH, model=0H, stepping<=09H), P6 family or older processors supporting MP operations (family=06H, extended_model=0, model<=0DH; or family<06H):
 - Each processor broadcasts a BIPI to "all including self." The first processor that broadcasts a BIPI (and thus receives its own BIPI vector), selects itself as the BSP and sets the BSP flag in its IA32_APIC_BASE MSR. (See Section 11.11.1, "Overview of the MP Initialization Process for P6 Family Processors," for a description of the BIPI, FIPI, and SIPI messages.)
 - The remainder of the processors (which were not selected as the BSP) are designated as APs. They leave their BSP flags in the clear state and enter a "wait-for-SIPI state."
 - The newly established BSP broadcasts an FIPI message to "all including self," which the BSP and APs treat as an end of MP initialization signal. Only the processor with its BSP flag set responds to the FIPI

message. It responds by fetching and executing the BIOS boot-strap code, beginning at the reset vector (physical address FFFF FFF0H).

5. As part of the boot-strap code, the BSP creates an ACPI table and/or an MP table and adds its initial APIC ID to these tables as appropriate.
6. At the end of the boot-strap procedure, the BSP sets a processor counter to 1, then broadcasts a SIPI message to all the APs in the system. Here, the SIPI message contains a vector to the BIOS AP initialization code (at 000VV000H, where VV is the vector contained in the SIPI message).
7. The first action of the AP initialization code is to set up a race (among the APs) to a BIOS initialization semaphore. The first AP to the semaphore begins executing the initialization code. (See Section 11.4.4, “MP Initialization Example,” for semaphore implementation details.) As part of the AP initialization procedure, the AP adds its APIC ID number to the ACPI and/or MP tables as appropriate and increments the processor counter by 1. At the completion of the initialization procedure, the AP executes a CLI instruction and halts itself.
8. When each of the APs has gained access to the semaphore and executed the AP initialization code, the BSP establishes a count for the number of processors connected to the system bus, completes executing the BIOS boot-strap code, and then begins executing operating-system boot-strap and start-up code.
9. While the BSP is executing operating-system boot-strap and start-up code, the APs remain in the halted state. In this state they will respond only to INITs, NMIs, and SMIs. They will also respond to snoops and to assertions of the STPCLK# pin.

The following section gives an example (with code) of the MP initialization protocol for of multiple processors operating in an MP configuration.

Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, describes how to program the LINT[0:1] pins of the processor’s local APICs after an MP configuration has been completed.

11.4.4 MP Initialization Example

The following example illustrates the use of the MP initialization protocol used to initialize processors in an MP system after the BSP and APs have been established. The code runs on Intel 64 or IA-32 processors that use a protocol. This includes P6 Family processors, Pentium 4 processors, Intel Core Duo, Intel Core 2 Duo and Intel Xeon processors.

The following constants and data definitions are used in the accompanying code examples. They are based on the addresses of the APIC registers defined in Table 13-1.

ICR_LOW	EQU OFEE00300H
SVR	EQU OFEE000F0H
APIC_ID	EQU OFEE00020H
LVT3	EQU OFEE00370H
APIC_ENABLED	EQU 0100H
BOOT_ID	DD ?
COUNT	EQU 00H
VACANT	EQU 00H

11.4.4.1 Typical BSP Initialization Sequence

After the BSP and APs have been selected (by means of a hardware protocol, see Section 11.4.3, “MP Initialization Protocol Algorithm for MP Systems”), the BSP begins executing BIOS boot-strap code (POST) at the normal IA-32 architecture starting address (FFFF FFF0H). The boot-strap code typically performs the following operations:

1. Initializes memory.
2. Loads the microcode update into the processor.
3. Initializes the MTRRs.
4. Enables the caches.

5. Executes the CUID instruction with a value of 0H in the EAX register, then reads the EBX, ECX, and EDX registers to determine if the BSP is "GenuineIntel."
6. Executes the CUID instruction with a value of 1H in the EAX register, then saves the values in the EAX, ECX, and EDX registers in a system configuration space in RAM for use later.
7. Loads start-up code for the AP to execute into a 4-KByte page in the lower 1 MByte of memory.
8. Switches to protected mode and ensures that the APIC address space is mapped to the strong uncacheable (UC) memory type.
9. Determine the BSP's APIC ID from the local APIC ID register (default is 0), the code snippet below is an example that applies to logical processors in a system whose local APIC units operate in xAPIC mode that APIC registers are accessed using memory mapped interface:

```
MOV ESI, APIC_ID; Address of local APIC ID register
MOV EAX, [ESI];
AND EAX, 0FF00000H; Zero out all other bits except APIC ID
MOV BOOT_ID, EAX; Save in memory
```

Saves the APIC ID in the ACPI and/or MP tables and optionally in the system configuration space in RAM.

10. Converts the base address of the 4-KByte page for the AP's bootup code into 8-bit vector. The 8-bit vector defines the address of a 4-KByte page in the real-address mode address space (1-MByte space). For example, a vector of 0BDH specifies a start-up memory address of 000BD000H.
11. Enables the local APIC by setting bit 8 of the APIC spurious vector register (SVR).

```
MOV ESI, SVR; Address of SVR
MOV EAX, [ESI];
OR EAX, APIC_ENABLED; Set bit 8 to enable (0 on reset)
MOV [ESI], EAX;
```

12. Sets up the LVT error handling entry by establishing an 8-bit vector for the APIC error handler.

```
MOV ESI, LVT3;
MOV EAX, [ESI];
AND EAX, 0FFFFFF0H; Clear out previous vector.
OR EAX, 000000xxH; xx is the 8-bit vector the APIC error handler.
MOV [ESI], EAX;
```

13. Initializes the Lock Semaphore variable VACANT to 00H. The APs use this semaphore to determine the order in which they execute BIOS AP initialization code.
14. Performs the following operation to set up the BSP to detect the presence of APs in the system and the number of processors (within a finite duration, minimally 100 milliseconds):
 - Sets the value of the COUNT variable to 1.
 - In the AP BIOS initialization code, the AP will increment the COUNT variable to indicate its presence. The finite duration while waiting for the COUNT to be updated can be accomplished with a timer. When the timer expires, the BSP checks the value of the COUNT variable. If the timer expires and the COUNT variable has not been incremented, no APs are present or some error has occurred.

15. Broadcasts an INIT-SIPI-SIPI IPI sequence to the APs to wake them up and initialize them. Alternatively, following a power-up or RESET, since all APs are already in the "wait-for-SIPI state," the BSP can broadcast just a single SIPI IPI to the APs to wake them up and initialize them. If software knows how many logical processors it expects to wake up, it may choose to poll the COUNT variable. If the expected processors show up before the 100 millisecond timer expires, the timer can be canceled and skip to step 16.

The left-hand-side of the procedure illustrated in Table 11-1 provides an algorithm when the expected processor count is unknown. The right-hand-side of Table 11-1 can be used when the expected processor count is known.

Table 11-1. Broadcast INIT-SIPI-SIPI Sequence and Choice of Timeouts

INIT-SIPI-SIPI when the expected processor count is unknown	INIT-SIPI-SIPI when the expected processor count is known
MOV ESI, ICR_LOW; Load address of ICR low dword into ESI. MOV EAX, 000C4500H; Load ICR encoding for broadcast INIT IPI ; to all APs into EAX. MOV [ESI], EAX; Broadcast INIT IPI to all APs ; 10-millisecond delay loop. MOV EAX, 000C46XXH; Load ICR encoding for broadcast SIPI IP ; to all APs into EAX, where xx is the vector computed in step 10. MOV [ESI], EAX; Broadcast SIPI IPI to all APs ; 200-microsecond delay loop MOV [ESI], EAX; Broadcast second SIPI IPI to all APs ; Waits for the timer interrupt until the timer expires	MOV ESI, ICR_LOW; Load address of ICR low dword into ESI. MOV EAX, 000C4500H; Load ICR encoding for broadcast INIT IPI ; to all APs into EAX. MOV [ESI], EAX; Broadcast INIT IPI to all APs ; 10-millisecond delay loop. MOV EAX, 000C46XXH; Load ICR encoding for broadcast SIPI IP ; to all APs into EAX, where xx is the vector computed in step 10. MOV [ESI], EAX; Broadcast SIPI IPI to all APs ; 200 microsecond delay loop with check to see if COUNT has ; reached the expected processor count. If COUNT reaches ; expected processor count, cancel timer and go to step 16. MOV [ESI], EAX; Broadcast second SIPI IPI to all APs ; Wait for the timer interrupt polling COUNT. If COUNT reaches ; expected processor count, cancel timer and go to step 16. ; If timer expires, go to step 16.

16. Reads and evaluates the COUNT variable and establishes a processor count.

17. If necessary, reconfigures the APIC and continues with the remaining system diagnostics as appropriate.

11.4.4.2 Typical AP Initialization Sequence

When an AP receives the SIPI, it begins executing BIOS AP initialization code at the vector encoded in the SIPI. The AP initialization code typically performs the following operations:

1. Waits on the BIOS initialization Lock Semaphore. When control of the semaphore is attained, initialization continues.
2. Loads the microcode update into the processor.
3. Initializes the MTRRs (using the same mapping that was used for the BSP).
4. Enables the cache.
5. Executes the CPUID instruction with a value of 0H in the EAX register, then reads the EBX, ECX, and EDX registers to determine if the AP is "GenuineIntel."
6. Executes the CPUID instruction with a value of 1H in the EAX register, then saves the values in the EAX, ECX, and EDX registers in a system configuration space in RAM for use later.
7. Switches to protected mode and ensures that the APIC address space is mapped to the strong uncacheable (UC) memory type.
8. Determines the AP's APIC ID from the local APIC ID register, and adds it to the MP and ACPI tables and optionally to the system configuration space in RAM.
9. Initializes and configures the local APIC by setting bit 8 in the SVR register and setting up the LVT3 (error LVT) for error handling (as described in steps 9 and 10 in Section 11.4.4.1, "Typical BSP Initialization Sequence").
10. Configures the APs SMI execution environment. (Each AP and the BSP must have a different SMBASE address.)
11. Increments the COUNT variable by 1.
12. Releases the semaphore.
13. Executes one of the following:

- the CLI and HLT instructions (if MONITOR/MWAIT is not supported), or
- the CLI, MONITOR, and MWAIT sequence to enter a deep C-state.

14. Waits for an INIT IPI.

11.4.5 Identifying Logical Processors in an MP System

After the BIOS has completed the MP initialization protocol, each logical processor can be uniquely identified by its local APIC ID. Software can access these APIC IDs in either of the following ways:

- **Read APIC ID for a local APIC** — Code running on a logical processor can read APIC ID in one of two ways depending on the local APIC unit is operating in x2APIC mode or in xAPIC mode:
 - If the local APIC unit supports x2APIC and is operating in x2APIC mode, 32-bit APIC ID can be read by executing a RDMSR instruction to read the processor's x2APIC ID register. This method is equivalent to executing CPUID.0BH described below.
 - If the local APIC unit is operating in xAPIC mode, 8-bit APIC ID can be read by executing a MOV instruction to read the processor's local APIC ID register (see Section 13.4.6, "Local APIC ID"). This is the ID to use for directing physical destination mode interrupts to the processor.
- **Read ACPI or MP table** — As part of the MP initialization protocol, the BIOS creates an ACPI table and an MP table. These tables are defined in the Multiprocessor Specification Version 1.4 and provide software with a list of the processors in the system and their local APIC IDs. The format of the ACPI table is derived from the ACPI specification, which is an industry standard power management and platform configuration specification for MP systems.
- **Read Initial APIC ID** (If the processor does not support CPUID.0BH) — An APIC ID is assigned to a logical processor during power up. This is the initial APIC ID reported by CPUID.01H:EBX[31:24] and may be different from the current value read from the local APIC. The initial APIC ID can be used to determine the topological relationship between logical processors for multi-processor systems that do not support CPUID.0BH.
 Bits in the 8-bit initial APIC ID can be interpreted using several bit masks. Each bit mask can be used to extract an identifier to represent a hierarchical domain of the multi-threading resource topology in an MP system (See Section 11.9.1, "Hierarchical Mapping of Shared Resources"). The initial APIC ID may consist of up to four bit-fields. In a non-clustered MP system, the field consists of up to three bit fields.
- **Read 32-bit APIC ID from CPUID.0BH** (If the processor supports CPUID.0BH) — A unique APIC ID is assigned to a logical processor during power up. This APIC ID is reported by CPUID.0BH:EDX[31:0] as a 32-bit value. Use the 32-bit APIC ID and CPUID.0BH to determine the topological relationship between logical processors if the processor supports CPUID.0BH.
 Bits in the 32-bit x2APIC ID can be extracted into sub-fields using CPUID.0BH parameters. (See Section 11.9.1, "Hierarchical Mapping of Shared Resources").

Figure 11-2 shows two examples of APIC ID bit fields in earlier single-core processors. In single-core Intel Xeon processors, the APIC ID assigned to a logical processor during power-up and initialization is 8 bits. Bits 2:1 form a 2-bit physical package identifier (which can also be thought of as a socket identifier). In systems that configure physical processors in clusters, bits 4:3 form a 2-bit cluster ID. Bit 0 is used in the Intel Xeon processor MP to identify the two logical processors within the package (see Section 11.9.3, "Hierarchical ID of Logical Processors in an MP System"). For Intel Xeon processors that do not support Intel Hyper-Threading Technology, bit 0 is always set to 0; for Intel Xeon processors supporting Intel Hyper-Threading Technology, bit 0 performs the same function as it does for Intel Xeon processor MP.

For more recent multi-core processors, see Section 11.9.1, "Hierarchical Mapping of Shared Resources," for a complete description of the topological relationships between logical processors and bit field locations within an initial APIC ID across Intel 64 and IA-32 processor families.

Note the number of bit fields and the width of bit-fields are dependent on processor and platform hardware capabilities. Software should determine these at runtime. When initial APIC IDs are assigned to logical processors, the value of APIC ID assigned to a logical processor will respect the bit-field boundaries corresponding core, physical package, etc. Additional examples of the bit fields in the initial APIC ID of multi-threading capable systems are shown in Section 11.9.

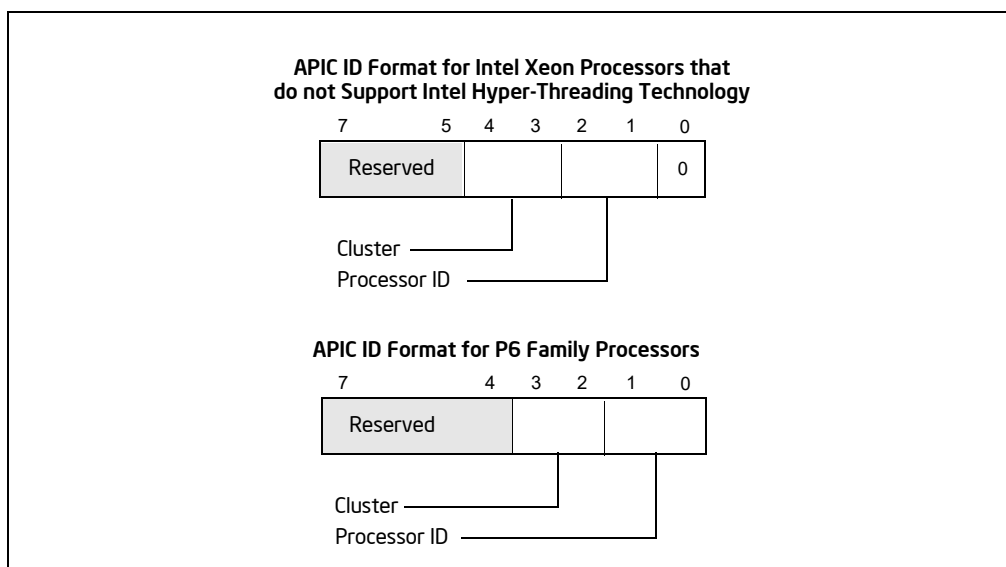


Figure 11-2. Interpretation of APIC ID in Early MP Systems

For P6 family processors, the APIC ID that is assigned to a processor during power-up and initialization is 4 bits (see Figure 11-2). Here, bits 0 and 1 form a 2-bit processor (or socket) identifier and bits 2 and 3 form a 2-bit cluster ID.

11.5 INTEL® HYPER-THREADING TECHNOLOGY AND INTEL® MULTI-CORE TECHNOLOGY

Intel Hyper-Threading Technology and Intel multi-core technology are extensions to Intel 64 and IA-32 architectures that enable a single physical processor to execute two or more separate code streams (called *threads*) concurrently. In Intel Hyper-Threading Technology, a single processor core provides two logical processors that share execution resources (see Section 11.7, “Intel® Hyper-Threading Technology Architecture”). In Intel multi-core technology, a physical processor package provides two or more processor cores. Both configurations require chipsets and a BIOS that support the technologies.

Software should not rely on processor names to determine whether a processor supports Intel Hyper-Threading Technology or Intel multi-core technology. Use the CPUID instruction to determine processor capability (see Section 11.6.2, “Initializing Multi-Core Processors”).

11.6 DETECTING HARDWARE MULTI-THREADING SUPPORT AND TOPOLOGY

Use the CPUID instruction to detect the presence of hardware multi-threading support in a physical processor. Hardware multi-threading can support several varieties of multigrade and/or Intel Hyper-Threading Technology. CPUID instruction provides several sets of parameter information to aid software enumerating topology information. The relevant topology enumeration parameters provided by CPUID include:

- **Hardware Multi-Threading feature flag (CPUID.01H:EDX[28] = 1)** — Indicates when set that the physical package is capable of supporting Intel Hyper-Threading Technology and/or multiple cores.
- **Processor topology enumeration parameters for 8-bit APIC ID:**
 - **Addressable IDs for Logical processors in the same Package (CPUID.01H:EBX[23:16])** — Indicates the maximum number of addressable ID for logical processors in a physical package. Within a physical package, there may be addressable IDs that are not occupied by any logical processors. This parameter does not represents the hardware capability of the physical processor.¹

- **Addressable IDs for processor cores in the same Package¹ (CPUID.04H.00H²:EAX[31:26] + 1 = Y)**
— Indicates the maximum number of addressable IDs attributable to processor cores (Y) in the physical package.
- **Extended Processor Topology Enumeration parameters for 32-bit APIC ID:** Intel 64 processors supporting CPUID.0BH will assign unique APIC IDs to each logical processor in the system. CPUID.0BH reports the 32-bit APIC ID and provide topology enumeration parameters. See CPUID instruction reference pages in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

The CPUID feature flag may indicate support for hardware multi-threading when only one logical processor available in the package. In this case, the decimal value represented by bits 16 through 23 in the EBX register will have a value of 1.

Software should note that the number of logical processors enabled by system software may be less than the value of "Addressable IDs for Logical processors". Similarly, the number of cores enabled by system software may be less than the value of "Addressable IDs for processor cores".

Software can detect the availability of the CPUID extended topology enumeration leaf (0BH) by performing two steps:

- Check maximum input value for basic CPUID information by using CPUID.00H. If CPUID.00H:EAX is greater than or equal to 11 (0BH), then proceed to next step,
- Check CPUID.0BH.00H:EBX is non-zero.

If both of the above conditions are true, extended topology enumeration leaf is available. Note the presence of CPUID.0BH in a processor does not guarantee support that the local APIC supports x2APIC. If CPUID.0BH.00H:EBX returns zero and maximum input value for basic CPUID information is greater than 0BH, then CPUID.0BH leaf is not supported on that processor.

11.6.1 Initializing Processors Supporting Intel® Hyper-Threading Technology

The initialization process for an MP system that contains processors supporting Intel Hyper-Threading Technology is the same as for conventional MP systems (see Section 11.4, "Multiple-Processor (MP) Initialization"). One logical processor in the system is selected as the BSP and other processors (or logical processors) are designated as APs. The initialization process is identical to that described in Section 11.4.3, "MP Initialization Protocol Algorithm for MP Systems," and Section 11.4.4, "MP Initialization Example."

During initialization, each logical processor is assigned an APIC ID that is stored in the local APIC ID register for each logical processor. If two or more processors supporting Intel Hyper-Threading Technology are present, each logical processor on the system bus is assigned a unique ID (see Section 11.9.3, "Hierarchical ID of Logical Processors in an MP System"). Once logical processors have APIC IDs, software communicates with them by sending APIC IPI messages.

11.6.2 Initializing Multi-Core Processors

The initialization process for an MP system that contains multi-core Intel 64 or IA-32 processors is the same as for conventional MP systems (see Section 11.4, "Multiple-Processor (MP) Initialization"). A logical processor in one core is selected as the BSP; other logical processors are designated as APs.

During initialization, each logical processor is assigned an APIC ID. Once logical processors have APIC IDs, software may communicate with them by sending APIC IPI messages.

-
1. Operating system and BIOS may implement features that reduce the number of logical processors available in a platform to applications at runtime to less than the number of physical packages times the number of hardware-capable logical processors per package.
 1. Software must check CPUID for its support of leaf 4 when implementing support for multi-core. If CPUID.04H is not available at runtime, software should handle the situation as if there is only one core per package.
 2. Maximum number of cores in the physical package must be queried by executing CPUID with EAX=4 and a valid ECX input value. Valid ECX input values start from 0.

11.6.3 Executing Multiple Threads on an Intel® 64 or IA-32 Processor Supporting Hardware Multi-Threading

Upon completing the operating system boot-up procedure, the bootstrap processor (BSP) executes operating system code. Other logical processors are placed in the halt state. To execute a code stream (thread) on a halted logical processor, the operating system issues an interprocessor interrupt (IPI) addressed to the halted logical processor. In response to the IPI, the processor wakes up and begins executing the code identified by the vector received as part of the IPI.

To manage execution of multiple threads on logical processors, an operating system can use conventional symmetric multiprocessing (SMP) techniques. For example, the operating-system can use a time-slice or load balancing mechanism to periodically interrupt each of the active logical processors. Upon interrupting a logical processor, the operating system checks its run queue for a thread waiting to be executed and dispatches the thread to the interrupted logical processor.

11.6.4 Handling Interrupts on an IA-32 Processor Supporting Hardware Multi-Threading

Interrupts are handled on processors supporting Intel Hyper-Threading Technology as they are on conventional MP systems. External interrupts are received by the I/O APIC, which distributes them as interrupt messages to specific logical processors (see Figure 11-3).

Logical processors can also send IPIs to other logical processors by writing to the ICR register of its local APIC (see Section 13.6, “Issuing Interprocessor Interrupts”). This also applies to dual-core processors.

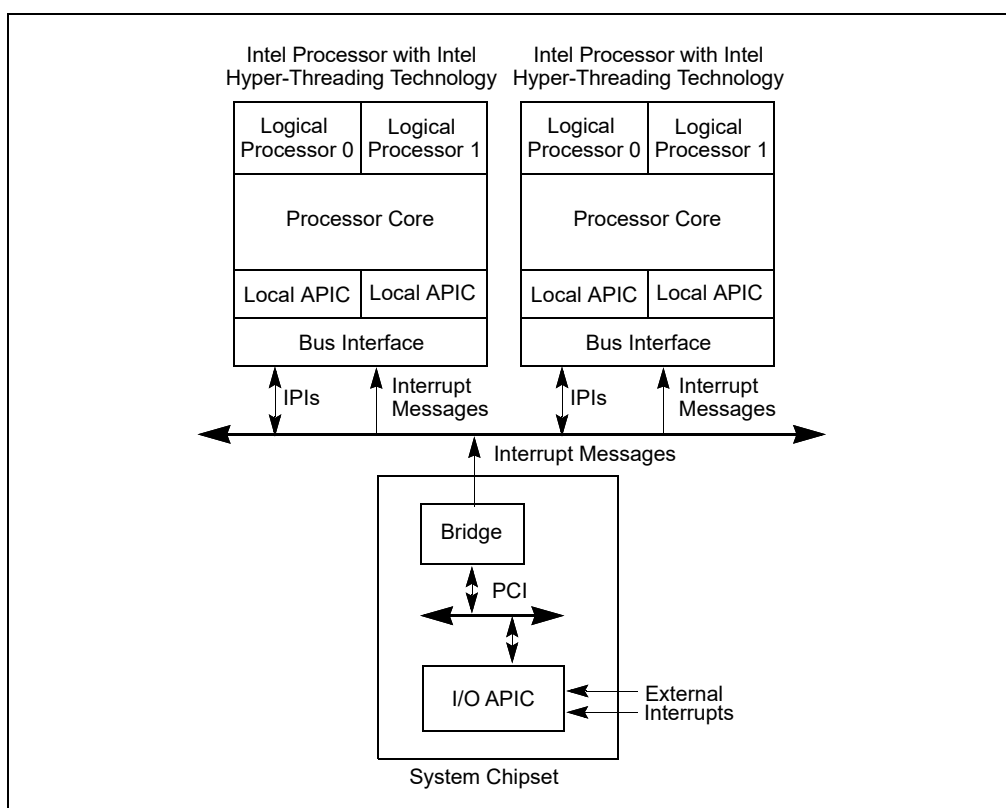


Figure 11-3. Local APICs and I/O APIC in MP System Supporting Intel HT Technology

11.7 INTEL® HYPER-THREADING TECHNOLOGY ARCHITECTURE

Figure 11-4 shows a generalized view of an Intel processor supporting Intel Hyper-Threading Technology, using the original Intel Xeon processor MP as an example. This implementation of the Intel Hyper-Threading Technology

consists of two logical processors (each represented by a separate architectural state) which share the processor's execution engine and the bus interface. Each logical processor also has its own advanced programmable interrupt controller (APIC).

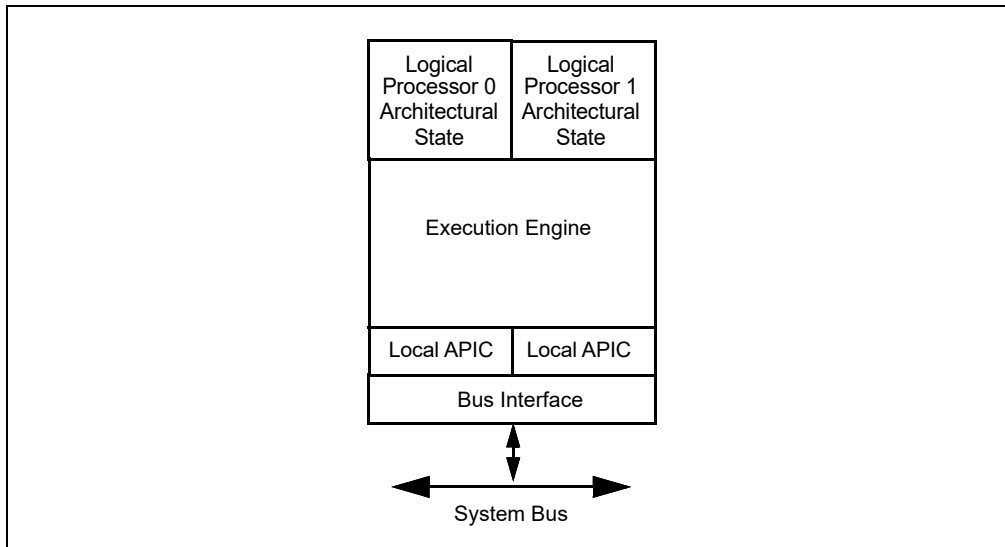


Figure 11-4. IA-32 Processor with Two Logical Processors Supporting Intel HT Technology

11.7.1 State of the Logical Processors

The following features are part of the architectural state of logical processors within Intel 64 or IA-32 processors supporting Intel Hyper-Threading Technology. The features can be subdivided into three groups:

- Duplicated for each logical processor
- Shared by logical processors in a physical processor
- Shared or duplicated, depending on the implementation

The following features are duplicated for each logical processor:

- General purpose registers (EAX, EBX, ECX, EDX, ESI, EDI, ESP, and EBP)
- Segment registers (CS, DS, SS, ES, FS, and GS)
- EFLAGS and EIP registers. Note that the CS and EIP/RIP registers for each logical processor point to the instruction stream for the thread being executed by the logical processor.
- x87 FPU registers (ST0 through ST7, status word, control word, tag word, data operand pointer, and instruction pointer)
- MMX registers (MM0 through MM7)
- XMM registers (XMM0 through XMM7) and the MXCSR register
- Control registers and system table pointer registers (GDTR, LDTR, IDTR, task register)
- Debug registers (DR0, DR1, DR2, DR3, DR6, DR7) and the debug control MSRs
- Machine check global status (IA32_MCG_STATUS) and machine check capability (IA32_MCG_CAP) MSRs
- Thermal clock modulation and ACPI Power management control MSRs
- Time stamp counter MSRs
- Most of the other MSR registers, including the page attribute table (PAT). See the exceptions below.
- Local APIC registers.
- Additional general purpose registers (R8-R15), XMM registers (XMM8-XMM15), control register, IA32_EFER on Intel 64 processors.

The following features are shared by logical processors:

- Memory type range registers (MTRRs)

Whether the following features are shared or duplicated is implementation-specific:

- IA32_MISC_ENABLE MSR (MSR address 1A0H)
- Machine check architecture (MCA) MSRs (except for the IA32_MCG_STATUS and IA32_MCG_CAP MSRs)
- Performance monitoring control and counter MSRs

11.7.2 APIC Functionality

When a processor supporting Intel Hyper-Threading Technology support is initialized, each logical processor is assigned a local APIC ID (see Table 13-1). The local APIC ID serves as an ID for the logical processor and is stored in the logical processor's APIC ID register. If two or more processors supporting Intel Hyper-Threading Technology are present in a dual processor (DP) or MP system, each logical processor on the system bus is assigned a unique local APIC ID (see Section 11.9.3, "Hierarchical ID of Logical Processors in an MP System").

Software communicates with local processors using the APIC's interprocessor interrupt (IPI) messaging facility. Setup and programming for APICs is identical in processors that support and do not support Intel Hyper-Threading Technology. See Chapter 13, "Advanced Programmable Interrupt Controller (APIC)," for a detailed discussion.

11.7.3 Memory Type Range Registers (MTRR)

MTRRs in a processor supporting Intel Hyper-Threading Technology are shared by logical processors. When one logical processor updates the setting of the MTRRs, settings are automatically shared with the other logical processors in the same physical package.

The architectures require that all MP systems based on Intel 64 and IA-32 processors (this includes logical processors) must use an identical MTRR memory map. This gives software a consistent view of memory, independent of the processor on which it is running. See Section 14.11, "Memory Type Range Registers (MTRRs)," for information on setting up MTRRs.

11.7.4 Page Attribute Table (PAT)

Each logical processor has its own PAT MSR (IA32_PAT). However, as described in Section 14.12, "Page Attribute Table (PAT)," the PAT MSR settings must be the same for all processors in a system, including the logical processors.

11.7.5 Machine Check Architecture

In the Intel HT Technology context as implemented by processors based on Intel NetBurst® microarchitecture, all of the machine check architecture (MCA) MSRs (except for the IA32_MCG_STATUS and IA32_MCG_CAP MSRs) are duplicated for each logical processor. This permits logical processors to initialize, configure, query, and handle machine-check exceptions simultaneously within the same physical processor. The design is compatible with machine check exception handlers that follow the guidelines given in Chapter 18, "Machine-Check Architecture."

The IA32_MCG_STATUS MSR is duplicated for each logical processor so that its machine check in progress bit field (MCIP) can be used to detect recursion on the part of MCA handlers. In addition, the MSR allows each logical processor to determine that a machine-check exception is in progress independent of the actions of another logical processor in the same physical package.

Because the logical processors within a physical package are tightly coupled with respect to shared hardware resources, both logical processors are notified of machine check errors that occur within a given physical processor. If machine-check exceptions are enabled when a fatal error is reported, all the logical processors within a physical package are dispatched to the machine-check exception handler. If machine-check exceptions are disabled, the logical processors enter the shutdown state and assert the IERR# signal.

When enabling machine-check exceptions, the MCE flag in control register CR4 should be set for each logical processor.

On Intel Atom family processors that support Intel Hyper-Threading Technology, the MCA facilities are shared between all logical processors on the same processor core.

11.7.6 Debug Registers and Extensions

Each logical processor has its own set of debug registers (DR0, DR1, DR2, DR3, DR6, DR7) and its own debug control MSR. These can be set to control and record debug information for each logical processor independently. Each logical processor also has its own last branch records (LBR) stack.

11.7.7 Performance Monitoring Counters

Performance counters and their companion control MSRs are shared between the logical processors within a processor core for processors based on Intel NetBurst microarchitecture. As a result, software must manage the use of these resources. The performance counter interrupts, events, and precise event monitoring support can be set up and allocated on a per thread (per logical processor) basis.

See Section 22.6.4, “Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture,” for a discussion of performance monitoring in the Intel Xeon processor MP.

In Intel Atom processor family that support Intel Hyper-Threading Technology, the performance counters (general-purpose and fixed-function counters) and their companion control MSRs are duplicated for each logical processor.

11.7.8 IA32_MISC_ENABLE MSR

The IA32_MISC_ENABLE MSR (MSR address 1A0H) is generally shared between the logical processors in a processor core supporting Intel Hyper-Threading Technology. However, some bit fields within IA32_MISC_ENABLE MSR may be duplicated per logical processor. The partition of shared or duplicated bit fields within IA32_MISC_ENABLE is implementation dependent. Software should program duplicated fields carefully on all logical processors in the system to ensure consistent behavior.

11.7.9 Memory Ordering

The logical processors in an Intel 64 or IA-32 processor supporting Intel Hyper-Threading Technology obey the same rules for memory ordering as Intel 64 or IA-32 processors without Intel HT Technology (see Section 11.2, “Memory Ordering”). Each logical processor uses a processor-ordered memory model that can be further defined as “write-ordered with store buffer forwarding.” All mechanisms for strengthening or weakening the memory-ordering model to handle special programming situations apply to each logical processor.

11.7.10 Serializing Instructions

As a general rule, when a logical processor in a processor supporting Intel Hyper-Threading Technology executes a serializing instruction, only that logical processor is affected by the operation. An exception to this rule is the execution of the WBINVD, INVD, and WRMSR instructions; and the MOV CR instruction when the state of the CD flag in control register CR0 is modified. Here, both logical processors are serialized.

11.7.11 Microcode Update Resources

In an Intel processor supporting Intel Hyper-Threading Technology, the microcode update facilities are shared between the logical processors; either logical processor can initiate an update. Each logical processor has its own BIOS signature MSR (IA32_BIOS_SIGN_ID at MSR address 8BH). When a logical processor performs an update for the physical processor, the IA32_BIOS_SIGN_ID MSRs for resident logical processors are updated with identical information. If logical processors initiate an update simultaneously, the processor core provides the necessary synchronization needed to ensure that only one update is performed at a time.

NOTE

Some processors (prior to the introduction of Intel 64 Architecture and based on Intel NetBurst microarchitecture) do not support simultaneous loading of microcode update to the sibling logical processors in the same core. All other processors support logical processors initiating an update simultaneously. Intel recommends a common approach that the microcode loader use the sequential technique described in Section 12.11.6.3.

11.7.12 Self Modifying Code

Intel processors supporting Intel Hyper-Threading Technology support self-modifying code, where data writes modify instructions cached or currently in flight. They also support cross-modifying code, where on an MP system writes generated by one processor modify instructions cached or currently in flight on another. See Section 11.1.3, “Handling Self- and Cross-Modifying Code,” for a description of the requirements for self- and cross-modifying code in an IA-32 processor.

11.7.13 Implementation-Specific Intel® HT Technology Facilities

The following non-architectural facilities are implementation-specific in IA-32 processors supporting Intel Hyper-Threading Technology:

- Caches.
- Translation lookaside buffers (TLBs).
- Thermal monitoring facilities.

The Intel Xeon processor MP implementation is described in the following sections.

11.7.13.1 Processor Caches

For processors supporting Intel Hyper-Threading Technology, the caches are shared. Any cache manipulation instruction that is executed on one logical processor has a global effect on the cache hierarchy of the physical processor. Note the following:

- **WBINVD instruction** — The entire cache hierarchy is invalidated after modified data is written back to memory. All logical processors are stopped from executing until after the write-back and invalidate operation is completed. A special bus cycle is sent to all caching agents. The amount of time or cycles for WBINVD to complete will vary due to the size of different cache hierarchies and other factors. As a consequence, the use of the WBINVD instruction can have an impact on interrupt/event response time.
- **INVD instruction** — The entire cache hierarchy is invalidated without writing back modified data to memory. All logical processors are stopped from executing until after the invalidate operation is completed. A special bus cycle is sent to all caching agents.
- **CLFLUSH and CLFLUSHOPT instructions** — The specified cache line is invalidated from the cache hierarchy after any modified data is written back to memory and a bus cycle is sent to all caching agents, regardless of which logical processor caused the cache line to be filled.
- **CD flag in control register CR0** — Each logical processor has its own CR0 control register, and thus its own CD flag in CR0. The CD flags for the two logical processors are ORed together, such that when any logical processor sets its CD flag, the entire cache is nominally disabled.

11.7.13.2 Processor Translation Lookaside Buffers (TLBs)

In processors supporting Intel Hyper-Threading Technology, data cache TLBs are shared. The instruction cache TLB may be duplicated or shared in each logical processor, depending on implementation specifics of different processor families.

Entries in the TLBs are tagged with an ID that indicates the logical processor that initiated the translation. This tag applies even for translations that are marked global using the page-global feature for memory paging. See Section 5.10, “Caching Translation Information,” for information about global translations.

When a logical processor performs a TLB invalidation operation, only the TLB entries that are tagged for that logical processor are guaranteed to be flushed. This protocol applies to all TLB invalidation operations, including writes to control registers CR3 and CR4 and uses of the INVLPG instruction.

11.7.13.3 Thermal Monitor

In a processor that supports Intel Hyper-Threading Technology, logical processors share the catastrophic shutdown detector and the automatic thermal monitoring mechanism (see Section 17.8, “Thermal Monitoring and Protection”). Sharing results in the following behavior:

- If the processor’s core temperature rises above the preset catastrophic shutdown temperature, the processor core halts execution, which causes both logical processors to stop execution.
- When the processor’s core temperature rises above the preset automatic thermal monitor trip temperature, the frequency of the processor core is automatically modulated, which effects the execution speed of both logical processors.

For software controlled clock modulation, each logical processor has its own IA32_CLOCK_MODULATION MSR, allowing clock modulation to be enabled or disabled on a logical processor basis. Typically, if software controlled clock modulation is going to be used, the feature must be enabled for all the logical processors within a physical processor and the modulation duty cycle must be set to the same value for each logical processor. If the duty cycle values differ between the logical processors, the processor clock will be modulated at the highest duty cycle selected.

11.7.13.4 External Signal Compatibility

This section describes the constraints on external signals received through the pins of a processor supporting Intel Hyper-Threading Technology and how these signals are shared between its logical processors.

- **STPCLK#** — A single STPCLK# pin is provided on the physical package of the Intel Xeon processor MP. External control logic uses this pin for power management within the system. When the STPCLK# signal is asserted, the processor core transitions to the stop-grant state, where instruction execution is halted but the processor core continues to respond to snoop transactions. Regardless of whether the logical processors are active or halted when the STPCLK# signal is asserted, execution is stopped on both logical processors and neither will respond to interrupts.

In MP systems, the STPCLK# pins on all physical processors are generally tied together. As a result this signal affects all the logical processors within the system simultaneously.

- **LINT0 and LINT1 pins** — A processor supporting Intel Hyper-Threading Technology has only one set of LINT0 and LINT1 pins, which are shared between the logical processors. When one of these pins is asserted, both logical processors respond unless the pin has been masked in the APIC local vector tables for one or both of the logical processors.

Typically in MP systems, the LINT0 and LINT1 pins are not used to deliver interrupts to the logical processors. Instead all interrupts are delivered to the local processors through the I/O APIC.

- **A20M# pin** — On an IA-32 processor, the A20M# pin is typically provided for compatibility with the Intel 286 processor. Asserting this pin causes bit 20 of the physical address to be masked (forced to zero) for all external bus memory accesses. Processors supporting Intel Hyper-Threading Technology provide one A20M# pin, which affects the operation of both logical processors within the physical processor.

The functionality of A20M# is used primarily by older operating systems and not used by modern operating systems. On newer Intel 64 processors, A20M# may be absent.

11.8 MULTI-CORE ARCHITECTURE

This section describes the architecture of Intel 64 and IA-32 processors supporting dual-core and quad-core technology. The discussion is applicable to the Intel Pentium processor Extreme Edition, Pentium D, Intel Core Duo, Intel Core 2 Duo, Dual-core Intel Xeon processor, Intel Core 2 Quad processors, and quad-core Intel Xeon processors. Features vary across different microarchitectures and are detectable using CPUID.

In general, each processor core has dedicated microarchitectural resources identical to a single-processor implementation of the underlying microarchitecture without hardware multi-threading capability. Each logical processor in a dual-core processor (whether supporting Intel Hyper-Threading Technology or not) has its own APIC functionality, PAT, machine check architecture, debug registers and extensions. Each logical processor handles serialization instructions or self-modifying code on its own. Memory order is handled the same way as in Intel Hyper-Threading Technology.

The topology of the cache hierarchy (with respect to whether a given cache level is shared by one or more processor cores or by all logical processors in the physical package) depends on the processor implementation. Software must use the deterministic cache parameter leaf of CPUID instruction to discover the cache-sharing topology between the logical processors in a multi-threading environment.

11.8.1 Logical Processor Support

The topological composition of processor cores and logical processors in a multi-core processor can be discovered using CPUID. Within each processor core, one or more logical processors may be available.

System software must follow the requirement MP initialization sequences (see Section 11.4, “Multiple-Processor (MP) Initialization”) to recognize and enable logical processors. At runtime, software can enumerate those logical processors enabled by system software to identify the topological relationships between these logical processors. (See Section 11.9.5, “Identifying Topological Relationships in an MP System”).

11.8.2 Memory Type Range Registers (MTRR)

MTRR is shared between two logical processors sharing a processor core if the physical processor supports Intel Hyper-Threading Technology. MTRR is not shared between logical processors located in different cores or different physical packages.

The Intel 64 and IA-32 architectures require that all logical processors in an MP system use an identical MTRR memory map. This gives software a consistent view of memory, independent of the processor on which it is running.

See Section 14.11, “Memory Type Range Registers (MTRRs).”

11.8.3 Performance Monitoring Counters

Performance counters and their companion control MSRs are shared between two logical processors sharing a processor core if the processor core supports Intel Hyper-Threading Technology and is based on Intel NetBurst microarchitecture. They are not shared between logical processors in different cores or different physical packages. As a result, software must manage the use of these resources, based on the topology of performance monitoring resources. Performance counter interrupts, events, and precise event monitoring support can be set up and allocated on a per thread (per logical processor) basis.

See Section 22.6.4, “Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture.”

11.8.4 IA32_MISC_ENABLE MSR

Some bit fields in IA32_MISC_ENABLE MSR (MSR address 1A0H) may be shared between two logical processors sharing a processor core, or may be shared between different cores in a physical processor. See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

11.8.5 Microcode Update Resources

Microcode update facilities are shared between two logical processors sharing a processor core if the physical package supports Intel Hyper-Threading Technology. They are not shared between logical processors in different

cores or different physical packages. Either logical processor that has access to the microcode update facility can initiate an update.

Each logical processor has its own BIOS signature MSR (IA32_BIOS_SIGN_ID at MSR address 8BH). When a logical processor performs an update for the physical processor, the IA32_BIOS_SIGN_ID MSRs for resident logical processors are updated with identical information.

All microcode update steps during processor initialization should use the same update data on all cores in all physical packages of the same stepping. Any subsequent microcode update must apply consistent update data to all cores in all physical packages of the same stepping. If the processor detects an attempt to load an older microcode update when a newer microcode update had previously been loaded, it may reject the older update to stay with the newer update.

NOTE

Some processors (prior to the introduction of Intel 64 Architecture and based on Intel NetBurst microarchitecture) do not support simultaneous loading of microcode update to the sibling logical processors in the same core. All other processors support logical processors initiating an update simultaneously. Intel recommends a common approach that the microcode loader use the sequential technique described in Section 12.11.6.3.

11.9 PROGRAMMING CONSIDERATIONS FOR HARDWARE MULTI-THREADING CAPABLE PROCESSORS

In a multi-threading environment, there may be certain hardware resources that are physically shared at some level of the hardware topology. In the multi-processor systems, typically bus and memory sub-systems are physically shared between multiple sockets. Within a hardware multi-threading capable processors, certain resources are provided for each processor core, while other resources may be provided for each logical processors (see Section 11.7, “Intel® Hyper-Threading Technology Architecture,” and Section 11.8, “Multi-Core Architecture”).

From a software programming perspective, control transfer of processor operation is managed at the granularity of logical processor (operating systems dispatch a runnable task by allocating an available logical processor on the platform). To manage the topology of shared resources in a multi-threading environment, it may be useful for software to understand and manage resources that are shared by more than one logical processors.

11.9.1 Hierarchical Mapping of Shared Resources

The APIC_ID value associated with each logical processor in a multi-processor system is unique (see Section 11.6, “Detecting Hardware Multi-Threading Support and Topology”). This 8-bit or 32-bit value can be decomposed into sub-fields, where each sub-field corresponds a hierarchical domain of the topological mapping of hardware resources.

The decomposition of an APIC_ID may consist of several sub fields representing the topology within a physical processor package, the higher-order bits of an APIC ID may also be used by cluster vendors to represent the topology of cluster nodes of each coherent multiprocessor systems:

- **Cluster** — Some multi-threading environments consists of multiple clusters of multi-processor systems. The CLUSTER_ID sub-field is usually supported by vendor firmware to distinguish different clusters. For non-clustered systems, CLUSTER_ID is usually 0 and system topology is reduced.
- **Package** — A physical processor package mates with a socket. A package may contain one or more software visible die. The PACKAGE_ID sub-field distinguishes different physical packages within a cluster.
- **Die** — A software-visible chip inside a package. The DIE_ID sub-field distinguishes different die within a package. If there are no software visible die, the width of this bit field is 0.
- **DieGrp** — A group of die that share certain resources.
- **Tile** — A set of cores that share certain resources. The TILE_ID sub-field distinguishes different tiles. If there are no software visible tiles, the width of this bit field is 0.

- **Module** — A set of cores that share certain resources. The MODULE_ID sub-field distinguishes different modules. If there are no software visible modules, the width of this bit field is 0.
- **Core** — Processor cores may be contained within modules, within tiles, on software-visible die, or appear directly at the package domain. The CORE_ID sub-field distinguishes processor cores. For a single-core processor, the width of this bit field is 0.
- **Logical Processor** — A processor core provides one or more logical processors sharing execution resources. The LOGICAL_PROCESSOR_ID sub-field distinguishes logical processors in a core. The width of this bit field is non-zero if a processor core provides more than one logical processors.

The LOGICAL_PROCESSOR_ID and CORE_ID sub-fields are bit-wise contiguous in the APIC_ID field (see Figure 11-5).

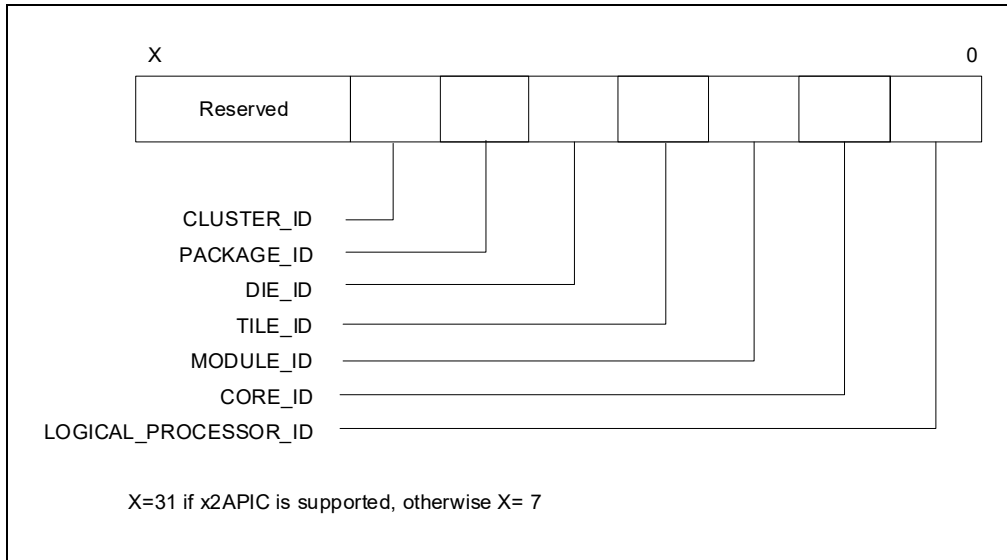


Figure 11-5. Generalized Seven-Domain Interpretation of the APIC ID

If the processor supports CPUID.0BH and CPUID.1FH, the 32-bit APIC ID can represent cluster plus several domains of topology within the physical processor package. The exact number of hierarchical domains within a physical processor package must be enumerated through CPUID.0BH and CPUID.1FH. Common processor families may employ a topology similar to that represented by the 8-bit Initial APIC ID. In general, CPUID.0BH and CPUID.1FH can support a topology enumeration algorithm that decompose a 32-bit APIC ID into more than four sub-fields (see Figure 11-6).

NOTE

CPUID.0BH and CPUID.1FH can have differences in the number of domain types reported (CPUID.1FH defines additional domain types). If the processor supports CPUID.1FH, usage of this leaf is preferred over CPUID.0BH. CPUID.0BH is available for legacy compatibility going forward.

The width of each sub-field depends on hardware and software configurations. Field widths can be determined at runtime using the algorithm discussed below (Example 11-16 through Example 11-21).

Figure 11-6 depicts the relationships of three of the hierarchical sub-fields in a hypothetical MP system. The value of valid APIC_IDs need not be contiguous across package boundary or core boundaries.

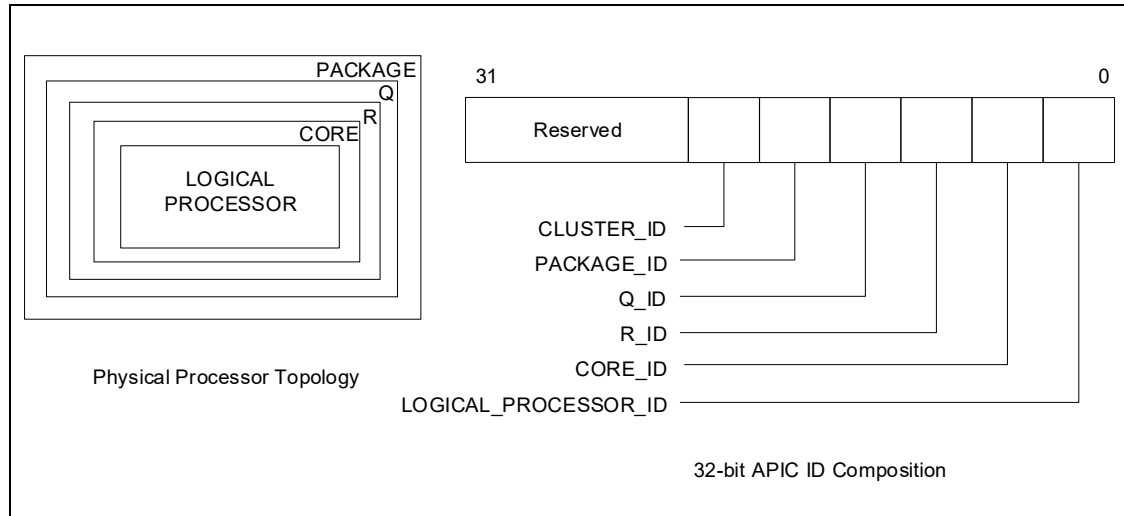


Figure 11-6. Conceptual Six-Domain Topology and 32-bit APIC ID Composition

11.9.2 Hierarchical Mapping of CPUID Extended Topology Leaf

CPUID.0BH and CPUID.1FH provide enumeration parameters for software to identify each hierarchy of the processor topology in a deterministic manner. Each hierarchical domain of the topology starting from the Logical Processor domain is represented numerically by a sub-leaf index within the CPUID 0BH leaf and 1FH leaf. Each domain of the topology is mapped to a sub-field in the APIC ID, following the general relationship depicted in Figure 11-6. This mechanism allows software to query the exact number of domains within a physical processor package and the bit-width of each sub-field of x2APIC ID directly. For example,

- Starting from sub-leaf index 0 and incrementing the input ECX, N, until the output ECX[15:8] returned by either CPUID.0BH.N or CPUID.1FH.N returns an invalid “domain type” encoding. The number of domains within the physical processor package is “N” (excluding PACKAGE). Using Figure 11-6 as an example, ECX[15:8] returned by either CPUID.0BH.04H or CPUID.1FH.04H will report 00H, indicating sub leaf 04H is invalid. This is also depicted by a pseudo code example:

Example 11-16. Number of Domains Below the Physical Processor Package

```
Word NumberOfDomainsBelowPackage = 0;
DWord Subleaf = 0;
```

```
EAX = 0BH or 1FH; // query each sub leaf of CPUID.0BH or 1FH; CPUID.1FH is preferred over CPUID.0BH if available.
```

```
ECX = Subleaf;
```

```
CPUID;
```

```
while (EBX != 0) // Enumerate until EBX reports 0
```

```
{
```

```
    if (EAX[4:0] != 0) // A Shift Value of 0 indicates this domain does not exist.
```

```
        // (Such as no SMT_ID, which is required entry at sub-leaf 0.)
```

```
    {
```

```
        NumberOfDomainsBelowPackage++;
```

```
    }
```

```
    Subleaf++;
```

```
    EAX = 0BH or 1FH;
```

```
    ECX = Subleaf;
```

```
    CPUID;
```

```
}
```

```
// NumberOfDomainsBelowPackage contains the absolute number of domains that exist below package.
```

N = Subleaf; // Sub-leaf supplies the number of entries CPUID will return.

- Sub-leaf index 0 (ECX = 0 as input) provides enumeration parameters to extract the LOGICAL_PROCESSOR_ID sub-field of x2APIC ID. If EAX[4:0] returned by either CPUID.0BH.00H or CPUID.1FH.00H reports a value (a right-shift count) that allow software to extract part of x2APIC ID to distinguish the next higher topological entities above the LOGICAL_PROCESSOR_ID domain. This value also corresponds to the bit-width of the sub-field of x2APIC ID corresponding the hierarchical domain with sub-leaf index 0.
- For each subsequent higher sub-leaf index m, EAX[4:0] returned by either CPUID.0BH.m or CPUID.1FH.m reports the right-shift count that will allow software to extract part of x2APIC ID to distinguish higher-domain topological entities. This means the right-shift value at of sub-leaf m, corresponds to the least significant (m+1) sub-fields of the 32-bit x2APIC ID.

Example 11-17. BitWidth Determination of x2APIC ID Sub-fields

```
For m = 0, m < N, m ++;
{cumulative_width[m] = CPUID.0BH.m:EAX[4:0] or CPUID.1FH.m:EAX[4:0]; }
BitWidth[0] = cumulative_width[0];
For m = 1, m < N, m ++;
    BitWidth[m] = cumulative_width[m] - cumulative_width[m-1];
```

NOTE

CPUID.1FH is a preferred superset to CPUID.0BH. CPUID.1FH defines additional domain types, and it must be parsed by an algorithm that can handle the addition of future domain types.

Previously, only the following encoding of hierarchical domain types were defined: 0 (invalid), 1 (logical processor), and 2 (core). With the additional hierarchical domain types available (see Section 11.9.1, “Hierarchical Mapping of Shared Resources,” and Figure 11-5, “Generalized Seven-Domain Interpretation of the APIC ID”) software must not assume any “domain type” encoding value to be related to any sub-leaf index, except sub-leaf 0.

Example 11-18. Support Routines for Identifying Package, Die, Core, and Logical Processors from 32-bit x2APIC ID

- a. **Derive the extraction bitmask for logical processors in a processor core and associated mask offset for different cores.**

```
//
// This example shows how to enumerate CPU topology domain types (domain types may or may not be known/supported by the
// software)
//
// Below is the list of sample domain types used in the example.
// Refer to the CPUID.1FH definition for the actual domain type numbers: “V2 Extended Topology Enumeration Leaf (Initial EAX Value
// = 1FH, ECX ≥ 0)”.
//
// LOGICAL PROCESSOR
// CORE
// MODULE
// TILE
// DIE
// PACKAGE
//
// The example shows how to identify and derive the extraction bitmask for the domains with identify type
// LOGICAL_PROCESSOR_ID/CORE_ID/DIE_ID/PACKAGE_ID
//
```

```
int DeriveLogical_Processor_Mask_Offsets (void)
{
```

```

IF (IHWMTSupported()) return -1;
execute cpuid with EAX = 0BH or 1FH, ECX = 0;
IF (returned domain type encoding in ECX[15:8] does not match LOGICAL_PROCESSOR_ID) return -1;
Mask_Logical_Processor_shift = EAX[4:0];    // # bits shift right of APIC ID to distinguish different cores, note this can be a shift
                                           // of zero if there is only one logical processor per core.
Logical Processor Mask = ~( (-1) << Mask_Logical_Processor_shift);    // shift left to derive extraction bitmask for
                                                                    // LOGICAL_PROCESSOR_ID
return 0;
}

```

b. Derive the extraction bitmask for processor cores in a physical processor package and associated mask offset for different packages.

```
int DeriveCore_Mask_Offsets (void)
```

```

{
    IF (IHWMTSupported()) return -1;
    execute cpuid with EAX = 0BH or 1FH, ECX = 0;
    WHILE ( ECX[15:8] ) {                // domain type encoding is valid
        Mask_last_known_shift = EAX[4:0]
        IF (returned domain type encoding in ECX[15:8] matches CORE) {
            Mask_Core_shift = EAX[4:0];
        }
        ELSE IF (returned domain type encoding in ECX[15:8] matches DIE {
            Mask_Die_shift = EAX[4:0];
        }
        //
        // Keep enumerating. Check if the next domain is the desired domain and if not, keep enumerating until you reach a known
        // domain or the invalid domain ("0" domain type). If there are more domains between DIE and PACKAGE, the unknown
        // domains will be ignored and treated as an extension of the last known domain (i.e., DIE in this case).
        //

        ECX++;
        execute cpuid with EAX = 0BH or 1FH;
    }

    COREPlusLogical_Processor_MASK = ~( (-1) << Mask_Core_shift);
    DIEPlusCORE_MASK = ~( (-1) << Mask_Die_shift);

    //
    // Treat domains between DIE and physical package as an extension of DIE for software choosing not to implement or recognize
    // these unknown domains.
    //

    CORE_MASK = COREPlusLogical_Processor_MASK ^ Logical Processor Mask;
    DIE_MASK = DIEPlusCORE_MASK ^ COREPlusLogical_Processor_MASK;
    PACKAGE_MASK = (-1) << Mask_last_known_shift;

    return -1;
}

```

11.9.3 Hierarchical ID of Logical Processors in an MP System

For Intel 64 and IA-32 processors, system hardware establishes an 8-bit initial APIC ID (or 32-bit APIC ID if the processor supports CPUID.0BH) that is unique for each logical processor following power-up or RESET (see Section 11.6.1). Each logical processor on the system is allocated an initial APIC ID. BIOS may implement features that tell the OS to support less than the total number of logical processors on the system bus. Those logical processors that are not available to applications at runtime are halted during the OS boot process. As a result, the number valid local APIC_IDs that can be queried by `affinitizing-current-thread-context` (See Example 11-23) is limited to the number of logical processors enabled at runtime by the OS boot process.

Table 11-2 shows an example of the 8-bit APIC IDs that are initially reported for logical processors in a system with four Intel Xeon MP processors that support Intel Hyper-Threading Technology (a total of 8 logical processors, each physical package has two processor cores and supports Intel Hyper-Threading Technology). Of the two logical processors within a Intel Xeon processor MP, logical processor 0 is designated the primary logical processor and logical processor 1 as the secondary logical processor.

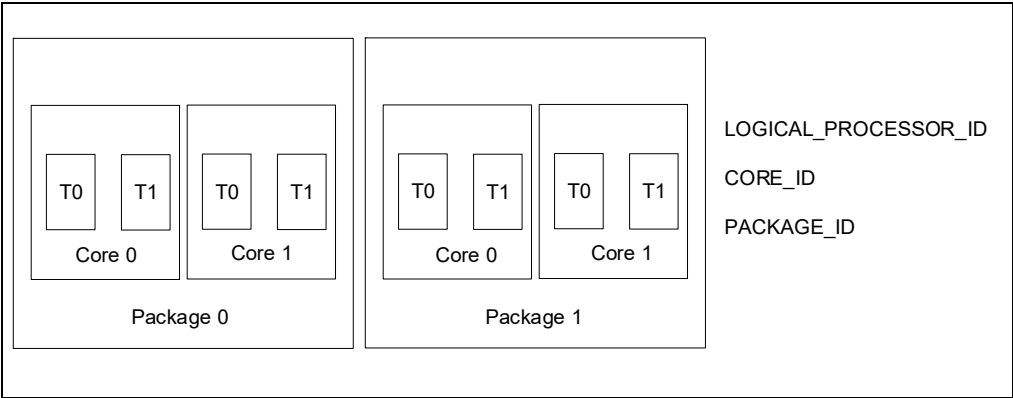


Figure 11-7. Topological Relationships Between Hierarchical IDs in a Hypothetical MP Platform

Table 11-2. Initial APIC IDs for the Logical Processors in a System that has Four Intel Xeon MP Processors Supporting Intel Hyper-Threading Technology¹

Initial APIC ID	PACKAGE_ID	CORE_ID	LOGICAL_PROCESSOR_ID
0H	0H	0H	0H
1H	0H	0H	1H
2H	1H	0H	0H
3H	1H	0H	1H
4H	2H	0H	0H
5H	2H	0H	1H
6H	3H	0H	0H
7H	3H	0H	1H

NOTE:
1. Because information on the number of processor cores in a physical package was not available in early single-core processors supporting Intel Hyper-Threading Technology, the CORE_ID can be treated as 0.

Table 11-3 shows the initial APIC IDs for a hypothetical situation with a dual processor system. Each physical package providing two processor cores, and each processor core also supporting Intel Hyper-Threading Technology.

Table 11-3. Initial APIC IDs for the Logical Processors in a System that has Two Physical Processors Supporting Dual-Core and Intel Hyper-Threading Technology

Initial APIC ID	PACKAGE_ID	CORE_ID	LOGICAL_PROCESSOR_ID
0H	0H	0H	0H
1H	0H	0H	1H
2H	0H	1H	0H
3H	0H	1H	1H
4H	1H	0H	0H
5H	1H	0H	1H
6H	1H	1H	0H
7H	1H	1H	1H

11.9.3.1 Hierarchical ID of Logical Processors with x2APIC ID

Table 11-4 shows an example of possible x2APIC ID assignments for a dual processor system that support x2APIC. Each physical package providing four processor cores, and each processor core also supporting Intel Hyper-Threading Technology. Note that the x2APIC ID need not be contiguous in the system.

Table 11-4. Example of Possible x2APIC ID Assignment in a System that has Two Physical Processors Supporting x2APIC and Intel Hyper-Threading Technology

x2APIC ID	PACKAGE_ID	CORE_ID	LOGICAL_PROCESSOR_ID
0H	0H	0H	0H
1H	0H	0H	1H
2H	0H	1H	0H
3H	0H	1H	1H
4H	0H	2H	0H
5H	0H	2H	1H
6H	0H	3H	0H
7H	0H	3H	1H
10H	1H	0H	0H
11H	1H	0H	1H
12H	1H	1H	0H
13H	1H	1H	1H
14H	1H	2H	0H
15H	1H	2H	1H
16H	1H	3H	0H
17H	1H	3H	1H

11.9.4 Algorithm for Three-Domain Mappings of APIC_ID

Software can gather the initial APIC_IDs for each logical processor supported by the operating system at runtime¹ and extract identifiers corresponding to the three domains of sharing topology (package, core, and logical processor). The three-domain algorithms below focus on a non-clustered MP system for simplicity. They do not assume APIC IDs are contiguous or that all logical processors on the platform are enabled.

Intel supports multi-threading systems where all physical processors report identical values in CPUID.0BH, CPUID.01H:EBX[23:16], CPUID.04H²:EAX[31:26], and CPUID.04H³:EAX[25:14]. The algorithms below assume the target system has symmetry across physical package boundaries with respect to the number of logical processors per package, number of cores per package, and cache topology within a package.

Software can choose to assume three-domain hierarchy if it was developed to understand only three domains. However, software implementation needs to ensure it does not break if it runs on systems that have more domains in the hierarchy even if it does not recognize them.

The extraction algorithm (for three-domain mappings from an APIC ID) uses the general procedure depicted in Example 11-19, and is supplemented by more detailed descriptions on the derivation of topology enumeration parameters for extraction bit masks:

1. Detect hardware multi-threading support in the processor.
2. Derive a set of bit masks that can extract the sub ID of each hierarchical domain of the topology. The algorithm to derive extraction bit masks for LOGICAL_PROCESSOR_ID/CORE_ID/PACKAGE_ID differs based on APIC ID is 32-bit (see step 3 below) or 8-bit (see step 4 below).
3. If the processor supports CPUID.0BH, each APIC ID contains a 32-bit value, the topology enumeration parameters needed to derive three-domain extraction bit masks are:
 - a. Query the right-shift value for the LOGICAL_PROCESSOR_ID domain of the topology using CPUID.0BH with ECX = 0H as input. The number of bits to shift-right on x2APIC ID (EAX[4:0]) can distinguish different higher-domain entities above logical processor in the same physical package. This is also the width of the bit mask to extract the LOGICAL_PROCESSOR_ID. The shift value may be 0 and enumerate no logical processor bit mask to create. A platform where cores only have one logical processor are not required to enumerate a separate bit layout for logical processor, and the lowest bits may only identify the core (where core and logical processor are then synonymous).
 - b. Enumerate until the desired domain is found (i.e., processor cores). Determine if the next domain is the expected domain. If the next domain is not known to the software, keep enumerating until the next known or the last domain. Software should use the previous domain before this to represent the last previously known domain (i.e., processor cores). If the software does not recognize or implement certain hierarchical domains, it should assume these unknown domains as an extension of the last known domain.
 - c. Query CPUID.0BH for the amount of bit shift to distinguish next higher-domain entities (e.g., physical processor packages) in the system. This describes an explicit three-domain-topology situation for commonly available processors. Consult Example 11-17 to adapt to situations beyond a three-domain topology of a physical processor. The width of the extraction bit mask can be used to derive the cumulative extraction bitmask to extract the sub IDs of logical processors (including different processor cores) in the same physical package. The extraction bit mask to distinguish merely different processor cores can be derived by xor'ing the logical processor extraction bit mask from the cumulative extraction bit mask.
 - d. Query the 32-bit x2APIC ID for the logical processor where the current thread is executing.
 - e. Derive the extraction bit masks corresponding to LOGICAL_PROCESSOR_ID, CORE_ID, and PACKAGE_ID, starting from LOGICAL_PROCESSOR_ID.
 - f. Apply each extraction bit mask to the 32-bit x2APIC ID to extract sub-field IDs.

-
1. As noted in Section 11.6 and Section 11.9.3, the number of logical processors supported by the OS at runtime may be less than the total number logical processors available in the platform hardware.
 2. Maximum number of addressable ID for processor cores in a physical processor is obtained by executing CPUID with EAX = 4 and a valid ECX index. The ECX index starts at 0.
 3. Maximum number addressable ID for processor cores sharing the target cache level is obtained by executing CPUID with EAX = 4 and the ECX index corresponding to the target cache level.

4. If the processor does not support CPUID.0BH, each initial APIC ID contains an 8-bit value, the topology enumeration parameters needed to derive extraction bit masks are:
 - a. Query the size of address space for sub IDs that can accommodate logical processors in a physical processor package. This size parameters (CPUID.01H:EBX[23:16]) can be used to derive the width of an extraction bitmask to enumerate the sub IDs of different logical processors in the same physical package.
 - b. Query the size of address space for sub IDs that can accommodate processor cores in a physical processor package. This size parameters can be used to derive the width of an extraction bitmask to enumerate the sub IDs of processor cores in the same physical package.
 - c. Query the 8-bit initial APIC ID for the logical processor where the current thread is executing.
 - d. Derive the extraction bit masks using respective address sizes corresponding to LOGICAL_PROCESSOR_ID, CORE_ID, and PACKAGE_ID, starting from LOGICAL_PROCESSOR_ID.
 - e. Apply each extraction bit mask to the 8-bit initial APIC ID to extract sub-field IDs.

Example 11-19. Support Routines for Detecting Hardware Multi-Threading and Identifying the Relationships Between Package, Core, and Logical Processors

1. Detect support for Hardware Multi-Threading Support in a processor.

```
// Returns a non-zero value if CPUID reports the presence of hardware multi-threading
// support in the physical package where the current logical processor is located.
// This does not guarantee BIOS or OS will enable all logical processors in the physical
// package and make them available to applications.
// Returns zero if hardware multi-threading is not present.
```

```
#define HWMT_BIT 10000000H
```

```
unsigned int HWMTSupported(void)
{
    // ensure cpuid instruction is supported
    execute cpuid with eax = 0 to get vendor string
    execute cpuid with eax = 1 to get feature flag and signature

    // Check to see if this a Genuine Intel Processor

    if (vendor string EQ GenuineIntel) {
        return (feature_flag_edx & HWMT_BIT); // bit 28
    }
    return 0;
}
```

Example 11-20. Support Routines for Identifying Package, Core, and Logical Processors from 32-bit x2APIC ID

a. Derive the extraction bitmask for logical processors in a processor core and associated mask offset for different cores.

```
int DeriveLogical_Processor_Mask_Offsets (void)
{
    if (!HWMTSupported()) return -1;
    execute cpuid with eax = 11, ECX = 0;
    If (returned domain type encoding in ECX[15:8] does not match logical processor) return -1;
    Mask_Logical_Processor_shift = EAX[4:0]; // # bits shift right of APIC ID to distinguish different cores, note this can be a shift
    // of zero if there is only one logical processor per core.
    Logical_Processor_Mask = ~( (-1) << Mask_Logical_Processor_shift); // shift left to derive extraction bitmask for
    // LOGICAL_PROCESSOR_ID
```

```
    return 0;
}
```

- b. Derive the extraction bitmask for processor cores in a physical processor package and associated mask offset for different packages.**

```
int DeriveCore_Mask_Offsets (void)
{
    if (!HWMTSupported()) return -1;
    execute cpuid with eax = 11, ECX = 0;
    while ( ECX[15:8] ) {          // domain type encoding is valid
        Mask_Core_shift = EAX[4:0];    // needed to distinguish different physical packages
        ECX ++;
        execute cpuid with eax = 11;
    }
    COREPlusLogical_Processor_MASK = ~( -1 ) << Mask_Core_shift;
    // treat domains between core and physical package as a core for software choosing not to implement or recognize
    // these unknown domains
    CORE_MASK = COREPlusLogical_Processor_MASK ^ Logical_Processor_Mask;
    PACKAGE_MASK = ( -1 ) << Mask_Core_shift;
    return -1;
}
```

- c. Query the x2APIC ID of a logical processor.**

APIC_IDs for each logical processor.

```
unsigned char Getx2APIC_ID (void)
{
    unsigned reg_edx = 0;
    execute cpuid with eax = 11, ECX = 0
    store returned value of edx
    return (unsigned) (reg_edx);
}
```

Example 11-21. Support Routines for Identifying Package, Core, and Logical Processors from 8-bit Initial APIC ID

- a. Find the size of address space for logical processors in a physical processor package.**

```
#define NUM_LOGICAL_BITS 00FF0000H
// Use the mask above and CPUID.01H:EBX[23:16] to obtain the max number of addressable IDs
// for logical processors in a physical package,

//Returns the size of address space of logical processors in a physical processor package;
// Software should not assume the value to be a power of 2.

unsigned char MaxLPIDsPerPackage(void)
{
    if (!HWMTSupported()) return 1;
    execute cpuid with eax = 1
    store returned value of ebx
    return (unsigned char) ((reg_ebx & NUM_LOGICAL_BITS) >> 16);
}
```

b. Find the size of address space for processor cores in a physical processor package.

// Returns the max number of addressable IDs for processor cores in a physical processor package;
 // Software should not assume cpuid reports this value to be a power of 2.

```
unsigned MaxCoreIDsPerPackage(void)
{
    if (!HWMTSupported()) return (unsigned char) 1;
    if cpuid supports leaf number 4
    { // we can retrieve multi-core topology info using leaf 4
        execute cpuid with eax = 4, ecx = 0
        store returned value of eax
        return (unsigned) ((reg_eax >> 26) + 1);
    }
    else // must be a single-core processor
        return 1;
}
```

c. Query the initial APIC ID of a logical processor.

```
#define INITIAL_APIC_ID_BITS FF000000H // CPUID.01H:EBX[31:24] initial APIC ID
```

// Returns the 8-bit unique initial APIC ID for the processor running the code.
 // Software can use OS services to affinitize the current thread to each logical processor
 // available under the OS to gather the initial APIC_IDs for each logical processor.

```
unsigned GetInitAPIC_ID (void)
{
    unsigned int reg_ebx = 0;
    execute cpuid with eax = 1
    store returned value of ebx
    return (unsigned) ((reg_ebx & INITIAL_APIC_ID_BITS) >> 24);
}
```

d. Find the width of an extraction bitmask from the maximum count of the bit-field (address size).

// Returns the mask bit width of a bit field from the maximum count that bit field can represent.
 // This algorithm does not assume 'address size' to have a value equal to power of 2.
 // Address size for LOGICAL_PROCESSOR_ID can be calculated from MaxLPIDsPerPackage()/MaxCoreIDsPerPackage()
 // Then use the routine below to derive the corresponding width of logical processor extraction bitmask
 // Address size for CORE_ID is MaxCoreIDsPerPackage(),
 // Derive the bitwidth for CORE extraction mask similarly

```
unsigned FindMaskWidth(unsigned Max_Count)
{unsigned int mask_width, cnt = Max_Count;
  __asm {
      mov eax, cnt
      mov ecx, 0
      mov mask_width, ecx
      dec eax
      bsr cx, ax
      jz next
      inc cx
      mov mask_width, ecx
      next:
      mov eax, mask_width
```

```

    }
    return mask_width;
}

```

e. Extract a sub ID from an 8-bit full ID, using address size of the sub ID and shift count.

```

// The routine below can extract LOGICAL_PROCESSOR_ID, CORE_ID, and PACKAGE_ID respectively from the init APIC_ID
// To extract LOGICAL_PROCESSOR_ID, MaxSubIDValue is set to the address size of LOGICAL_PROCESSOR_ID, Shift_Count = 0
// To extract CORE_ID, MaxSubIDValue is the address size of CORE_ID, Shift_Count is width of logical processor extraction bitmask.
// Returns the value of the sub ID, this is not a zero-based value

```

```

Unsigned char GetSubID(unsigned char Full_ID, unsigned char MaxSubIDValue, unsigned char Shift_Count)
{
    MaskWidth = FindMaskWidth(MaxSubIDValue);
    MaskBits = ((uchar) (FFH << Shift_Count)) ^ ((uchar) (FFH << Shift_Count + MaskWidth));
    SubID = Full_ID & MaskBits;
    Return SubID;
}

```

Software must not assume local APIC_ID values in an MP system are consecutive. Non-consecutive local APIC_IDs may be the result of hardware configurations or debug features implemented in the BIOS or OS.

An identifier for each hierarchical domain can be extracted from an 8-bit APIC_ID using the support routines illustrated in Example 11-21. The appropriate bit mask and shift value to construct the appropriate bit mask for each domain must be determined dynamically at runtime.

11.9.5 Identifying Topological Relationships in an MP System

To detect the number of physical packages, processor cores, or other topological relationships in a MP system, the following procedures are recommended:

- Extract the three-domain identifiers from the APIC ID of each logical processor enabled by system software. The sequence is as follows (see the pseudo code shown in Example 11-22 and support routines shown in Example 11-19):
 - The extraction start from the right-most bit field, corresponding to LOGICAL_PROCESSOR_ID, the innermost hierarchy in a three-domain topology (See Figure 11-7). For the right-most bit field, the shift value of the working mask is zero. The width of the bit field is determined dynamically using the maximum number of logical processor per core, which can be derived from information provided from CPUID.
 - To extract the next bit-field, the shift value of the working mask is determined from the width of the bit mask of the previous step. The width of the bit field is determined dynamically using the maximum number of cores per package.
 - To extract the remaining bit-field, the shift value of the working mask is determined from the maximum number of logical processor per package. So the remaining bits in the APIC ID (excluding those bits already extracted in the two previous steps) are extracted as the third identifier. This applies to a non-clustered MP system, or if there is no need to distinguish between PACKAGE_ID and CLUSTER_ID.

If there is need to distinguish between PACKAGE_ID and CLUSTER_ID, PACKAGE_ID can be extracted using an algorithm similar to the extraction of CORE_ID, assuming the number of physical packages in each node of a clustered system is symmetric.
- Assemble the three-domain identifiers of LOGICAL_PROCESSOR_ID, CORE_ID, PACKAGE_IDs into arrays for each enabled logical processor. This is shown in Example 11-23a.
- To detect the number of physical packages: use PACKAGE_ID to identify those logical processors that reside in the same physical package. This is shown in Example 11-23b. This example also depicts a technique to construct a mask to represent the logical processors that reside in the same package.

- To detect the number of processor cores: use CORE_ID to identify those logical processors that reside in the same core. This is shown in Example 11-23. This example also depicts a technique to construct a mask to represent the logical processors that reside in the same core.

In Example 11-22, the numerical ID value can be obtained from the value extracted with the mask by shifting it right by shift count. Algorithms below do not shift the value. The assumption is that the SubID values can be compared for equivalence without the need to shift.

Example 11-22. Pseudo Code Depicting Three-Domain Extraction Algorithm

```

For Each local_APIC_ID{
    // Calculate Logical Processor Mask, the bit mask pattern to extract LOGICAL_PROCESSOR_ID,
    // Logical Processor Mask is determined using topology enumeration parameters
    // from CPUID.0BH (Example 11-20);
    // otherwise, Logical Processor Mask is determined using CPUID.01H and leaf 04H (Example 11-21).
    // This algorithm assumes there is symmetry across core boundary, i.e., each core within a
    // package has the same number of logical processors
    // LOGICAL_PROCESSOR_ID always starts from bit 0, corresponding to the right-most bit-field
    LOGICAL_PROCESSOR_ID = APIC_ID & Logical Processor Mask;

    // Extract CORE_ID:
    // Core Mask is determined in Example 11-20 or Example 11-21
    CORE_ID = (APIC_ID & Core Mask);

    // Extract PACKAGE_ID:
    // Assume single cluster.
    // Shift out the mask width for maximum logical processors per package
    // Package Mask is determined in Example 11-20 or Example 11-21
    PACKAGE_ID = (APIC_ID & Package Mask);
}

```

Example 11-23. Compute the Number of Packages, Cores, and Processor Relationships in a MP System

a) Assemble lists of PACKAGE_ID, CORE_ID, and LOGICAL_PROCESSOR_ID of each enabled logical processors

```

// The BIOS and/or OS may limit the number of logical processors available to applications after system boot.
// The below algorithm will compute topology for the processors visible to the thread that is computing it.

// Extract the 3-domains of IDs on every processor.
// SystemAffinity is a bitmask of all the processors started by the OS. Use OS specific APIs to obtain it.
// ThreadAffinityMask is used to affinitize the topology enumeration thread to each processor using OS specific APIs.
// Allocate per processor arrays to store the Package_ID, Core_ID, and LOGICAL_PROCESSOR_ID for every started processor.

ThreadAffinityMask = 1;
ProcessorNum = 0;
while (ThreadAffinityMask ≠ 0 && ThreadAffinityMask ≤ SystemAffinity) {
    // Check to make sure we can utilize this processor first.
    if (ThreadAffinityMask & SystemAffinity){
        Set thread to run on the processor specified in ThreadAffinityMask
        Wait if necessary and ensure thread is running on specified processor

        APIC_ID = GetAPIC_ID(); // 32 bit ID in Example 11-20 or 8-bit ID in Example 11-21
        Extract the Package_ID, Core_ID, and LOGICAL_PROCESSOR_ID as explained in three domain extraction
        algorithm of Example 11-22
        PackageID[ProcessorNUM] = PACKAGE_ID;
        CoreID[ProcessorNum] = CORE_ID;
    }
    ThreadAffinityMask = ThreadAffinityMask - 1;
    ProcessorNum++;
}

```

```

        LOGICAL_PROCESSOR_ID[ProcessorNum] = LOGICAL_PROCESSOR_ID;
        ProcessorNum++;
    }
    ThreadAffinityMask <<= 1;
}
NumStartedLPs = ProcessorNum;

```

b) Using the list of PACKAGE_ID to count the number of physical packages in a MP system and construct, for each package, a multi-bit mask corresponding to those logical processors residing in the same package.

```

// Compute the number of packages by counting the number of processors with unique PACKAGE_IDs in the PackageID array.
// Compute the mask of processors in each package.

// PackageIDBucket is an array of unique PACKAGE_ID values. Allocate an array of NumStartedLPs count of entries in this array.
// PackageProcessorMask is a corresponding array of the bit mask of processors belonging to the same package, these are
// processors with the same PACKAGE_ID.
// The algorithm below assumes there is symmetry across package boundary if more than one socket is populated in an MP
//system.
// Bucket Package IDs and compute processor mask for every package.

PackageNum = 1;
PackageIDBucket[0] = PackageID[0];
ProcessorMask = 1;
PackageProcessorMask[0] = ProcessorMask;
For (ProcessorNum = 1; ProcessorNum < NumStartedLPs; ProcessorNum++) {
    ProcessorMask <<= 1;
    For (i=0; i < PackageNum; i++) {
        // we may be comparing bit-fields of logical processors residing in different
        // packages, the code below assume package symmetry
        If (PackageID[ProcessorNum] = PackageIDBucket[i]) {
            PackageProcessorMask[i] |= ProcessorMask;
            Break; // found in existing bucket, skip to next iteration
        }
    }
    if (i = PackageNum) {
        //PACKAGE_ID did not match any bucket, start new bucket
        PackageIDBucket[i] = PackageID[ProcessorNum];
        PackageProcessorMask[i] = ProcessorMask;
        PackageNum++;
    }
}
// PackageNum has the number of Packages started in OS
// PackageProcessorMask[] array has the processor set of each package

```

c) Using the list of CORE_ID to count the number of cores in a MP system and construct, for each core, a multi-bit mask corresponding to those logical processors residing in the same core.

Processors in the same core can be determined by bucketing the processors with the same PACKAGE_ID and CORE_ID. Note that code below can BIT OR the values of PACKGE and CORE ID because they have not been shifted right.

The algorithm below assumes there is symmetry across package boundary if more than one socket is populated in an MP system.

```

//Bucketing PACKAGE and CORE IDs and computing processor mask for every core
CoreNum = 1;
CoreIDBucket[0] = PackageID[0] | CoreID[0];
ProcessorMask = 1;

```



```

CoreProcessorMask[0] = ProcessorMask;
For (ProcessorNum = 1; ProcessorNum < NumStartedLPs; ProcessorNum++) {
    ProcessorMask <<= 1;
    For (i=0; i < CoreNum; i++) {
        // we may be comparing bit-fields of logical processors residing in different
        // packages, the code below assume package symmetry
        If ((PackageID[ProcessorNum] | CoreID[ProcessorNum]) = CoreIDBucket[i]) {
            CoreProcessorMask[i] |= ProcessorMask;
            Break; // found in existing bucket, skip to next iteration
        }
    }
    if (i = CoreNum) {
        //Did not match any bucket, start new bucket
        CoreIDBucket[i] = PackageID[ProcessorNum] | CoreID[ProcessorNum];
        CoreProcessorMask[i] = ProcessorMask;
        CoreNum++;
    }
}
// CoreNum has the number of cores started in the OS
// CoreProcessorMask[] array has the processor set of each core

```

Other processor relationships such as processor mask of sibling cores can be computed from set operations of the PackageProcessorMask[] and CoreProcessorMask[].

The algorithm shown above can be adapted to work with earlier generations of single-core IA-32 processors that support Intel Hyper-Threading Technology and in situations that the deterministic cache parameter leaf is not supported (provided CPUID supports initial APIC ID). A reference code example is available (see Intel® 64 Architecture Processor Topology Enumeration Technical Paper).

11.10 MANAGEMENT OF IDLE AND BLOCKED CONDITIONS

When a logical processor in an MP system (including multi-core processor or processors supporting Intel Hyper-Threading Technology) is idle (no work to do) or blocked (on a lock or semaphore), additional management of the core execution engine resource can be accomplished by using the HLT (halt), PAUSE, or the MONITOR/MWAIT instructions.

11.10.1 HLT Instruction

The HLT instruction stops the execution of the logical processor on which it is executed and places it in a halted state until further notice (see the description of the HLT instruction in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A). When a logical processor is halted, active logical processors continue to have full access to the shared resources within the physical package. Here shared resources that were being used by the halted logical processor become available to active logical processors, allowing them to execute at greater efficiency. When the halted logical processor resumes execution, shared resources are again shared among all active logical processors. (See Section 11.10.6.3, "Halt Idle Logical Processors," for more information about using the HLT instruction with processors supporting Intel Hyper-Threading Technology.)

11.10.2 PAUSE Instruction

The PAUSE instruction can improve the performance of processors supporting Intel Hyper-Threading Technology when executing "spin-wait loops" and other routines where one thread is accessing a shared lock or semaphore in a tight polling loop. When executing a spin-wait loop, the processor can suffer a severe performance penalty when exiting the loop because it detects a possible memory order violation and flushes the core processor's pipeline. The PAUSE instruction provides a hint to the processor that the code sequence is a spin-wait loop. The processor uses

this hint to avoid the memory order violation and prevent the pipeline flush. In addition, the PAUSE instruction depipelined the spin-wait loop to prevent it from consuming execution resources excessively and consume power needlessly. (See Section 11.10.6.1, “Use the PAUSE Instruction in Spin-Wait Loops,” for more information about using the PAUSE instruction with IA-32 processors supporting Intel Hyper-Threading Technology.)

11.10.3 Detecting Support MONITOR/MWAIT Instruction

Streaming SIMD Extensions 3 introduced two instructions (MONITOR and MWAIT) to help multithreaded software improve thread synchronization. In the initial implementation, MONITOR and MWAIT are available to software at ring 0. The instructions are conditionally available at levels greater than 0. Use the following steps to detect the availability of MONITOR and MWAIT:

- Use CPUID to query the MONITOR bit (CPUID.01H:ECX[3] = 1).
- If CPUID indicates support, execute MONITOR inside a TRY/EXCEPT exception handler and trap for an exception. If an exception occurs, MONITOR and MWAIT are not supported at a privilege level greater than 0. See Example 11-24.

Example 11-24. Verifying MONITOR/MWAIT Support

```
boolean MONITOR_MWAIT_works = TRUE;
try {
    _asm {
        xor ecx, ecx
        xor edx, edx
        mov eax, MemArea
        monitor
    }
    // Use monitor
} except (UNWIND) {
    // if we get here, MONITOR/MWAIT is not supported
    MONITOR_MWAIT_works = FALSE;
}
```

11.10.4 MONITOR/MWAIT Instruction

Operating systems usually implement idle loops to handle thread synchronization. In a typical idle-loop scenario, there could be several “busy loops” and they would use a set of memory locations. An impacted processor waits in a loop and poll a memory location to determine if there is available work to execute. The posting of work is typically a write to memory (the work-queue of the waiting processor). The time for initiating a work request and getting it scheduled is on the order of a few bus cycles.

From a resource sharing perspective (logical processors sharing execution resources), use of the HLT instruction in an OS idle loop is desirable but has implications. Executing the HLT instruction on a idle logical processor puts the targeted processor in a non-execution state. This requires another processor (when posting work for the halted logical processor) to wake up the halted processor using an inter-processor interrupt. The posting and servicing of such an interrupt introduces a delay in the servicing of new work requests.

In a shared memory configuration, exits from busy loops usually occur because of a state change applicable to a specific memory location; such a change tends to be triggered by writes to the memory location by another agent (typically a processor).

MONITOR/MWAIT complement the use of HLT and PAUSE to allow for efficient partitioning and un-partitioning of shared resources among logical processors sharing physical resources. MONITOR sets up an effective address range that is monitored for write-to-memory activities; MWAIT places the processor in an optimized state (this may vary between different implementations) until a write to the monitored address range occurs.

In the initial implementation of MONITOR and MWAIT, they are available at CPL = 0 only.

Both instructions rely on the state of the processor's monitor hardware. The monitor hardware can be either armed (by executing the MONITOR instruction) or triggered (due to a variety of events, including a store to the monitored memory region). If upon execution of MWAIT, monitor hardware is in a triggered state: MWAIT behaves as a NOP and execution continues at the next instruction in the execution stream. The state of monitor hardware is not architecturally visible except through the behavior of MWAIT.

Multiple events other than a write to the triggering address range can cause a processor that executed MWAIT to wake up. These include events that would lead to voluntary or involuntary context switches, such as:

- External interrupts, including NMI, SMI, INIT, BINIT, MCERR, A20M#
- Faults, Aborts (including Machine Check)
- Architectural TLB invalidations including writes to CR0, CR3, CR4, and certain MSR writes; execution of LMSW (occurring prior to issuing MWAIT but after setting the monitor)
- Voluntary transitions due to fast system call and far calls (occurring prior to issuing MWAIT but after setting the monitor)

Power management related events (such as Thermal Monitor 2 or chipset driven STPCLK# assertion) will not cause the monitor event pending flag to be cleared. Faults will not cause the monitor event pending flag to be cleared.

Software should not allow for voluntary context switches in between MONITOR/MWAIT in the instruction flow. Note that execution of MWAIT does not re-arm the monitor hardware. This means that MONITOR/MWAIT need to be executed in a loop. Also note that exits from the MWAIT state could be due to a condition other than a write to the triggering address; software should explicitly check the triggering data location to determine if the write occurred. Software should also check the value of the triggering address following the execution of the monitor instruction (and prior to the execution of the MWAIT instruction). This check is to identify any writes to the triggering address that occurred during the course of MONITOR execution.

The address range provided to the MONITOR instruction must be of write-back caching type. Only write-back memory type stores to the monitored address range will trigger the monitor hardware. If the address range is not in memory of write-back type, the address monitor hardware may not be set up properly or the monitor hardware may not be armed. Software is also responsible for ensuring that

- Writes that are not intended to cause the exit of a busy loop do not write to a location within the address region being monitored by the monitor hardware,
- Writes intended to cause the exit of a busy loop are written to locations within the monitored address region.

Not doing so will lead to more false wakeups (an exit from the MWAIT state not due to a write to the intended data location). These have negative performance implications. It might be necessary for software to use padding to prevent false wakeups. CPUID provides a mechanism for determining the size data locations for monitoring as well as a mechanism for determining the size of a the pad.

11.10.5 Monitor/Mwait Address Range Determination

To use the MONITOR/MWAIT instructions, software should know the length of the region monitored by the MONITOR/MWAIT instructions and the size of the coherence line size for cache-snoop traffic in a multiprocessor system. This information can be queried using CPUID.05H). You will need the smallest and largest monitor line size:

- To avoid missed wake-ups: make sure that the data structure used to monitor writes fits within the smallest monitor line-size. Otherwise, the processor may not wake up after a write intended to trigger an exit from MWAIT.
- To avoid false wake-ups; use the largest monitor line size to pad the data structure used to monitor writes. Software must make sure that beyond the data structure, no unrelated data variable exists in the triggering area for MWAIT. A pad may be needed to avoid this situation.

These above two values bear no relationship to cache line size in the system and software should not make any assumptions to that effect. Within a single-cluster system, the two parameters should default to be the same (the size of the monitor triggering area is the same as the system coherence line size).

Based on the monitor line sizes returned by the CPUID, the OS should dynamically allocate structures with appropriate padding. If static data structures must be used by an OS, attempt to adapt the data structure and use a

dynamically allocated data buffer for thread synchronization. When the latter technique is not possible, consider not using MONITOR/MWAIT when using static data structures.

To set up the data structure correctly for MONITOR/MWAIT on multi-clustered systems: interaction between processors, chipsets, and the BIOS is required (system coherence line size may depend on the chipset used in the system; the size could be different from the processor's monitor triggering area). The BIOS is responsible to set the correct value for system coherence line size using the IA32_MONITOR_FILTER_LINE_SIZE MSR. Depending on the relative magnitude of the size of the monitor triggering area versus the value written into the IA32_MONITOR_FILTER_LINE_SIZE MSR, the smaller of the parameters will be reported as the *Smallest Monitor Line Size*. The larger of the parameters will be reported as the *Largest Monitor Line Size*.

11.10.6 Required Operating System Support

This section describes changes that must be made to an operating system to run on processors supporting Intel Hyper-Threading Technology. It also describes optimizations that can help an operating system make more efficient use of the logical processors sharing execution resources. The required changes and suggested optimizations are representative of the types of modifications that appear in Windows* XP and Linux* kernel 2.4.0 operating systems for Intel processors supporting Intel Hyper-Threading Technology. Additional optimizations for processors supporting Intel Hyper-Threading Technology are described in the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

11.10.6.1 Use the PAUSE Instruction in Spin-Wait Loops

Intel recommends that a PAUSE instruction be placed in all spin-wait loops that run on Intel processors supporting Intel Hyper-Threading Technology and multi-core processors.

Software routines that use spin-wait loops include multiprocessor synchronization primitives (spin-locks, semaphores, and mutex variables) and idle loops. Such routines keep the processor core busy executing a load-compare-branch loop while a thread waits for a resource to become available. Including a PAUSE instruction in such a loop greatly improves efficiency (see Section 11.10.2, "PAUSE Instruction"). The following routine gives an example of a spin-wait loop that uses a PAUSE instruction:

```
Spin_Lock:
    CMP lockvar, 0    ;Check if lock is free
    JE Get_Lock
    PAUSE             ;Short delay
    JMP Spin_Lock
Get_Lock:
    MOV EAX, 1
    XCHG EAX, lockvar ;Try to get lock
    CMP EAX, 0        ;Test if successful
    JNE Spin_Lock
Critical_Section:
    <critical section code>
    MOV lockvar, 0
    ...
Continue:
```

The spin-wait loop above uses a "test, test-and-set" technique for determining the availability of the synchronization variable. This technique is recommended when writing spin-wait loops.

In IA-32 processor generations earlier than the Pentium 4 processor, the PAUSE instruction is treated as a NOP instruction.

11.10.6.2 Potential Usage of MONITOR/MWAIT in C0 Idle Loops

An operating system may implement different handlers for different idle states. A typical OS idle loop on an ACPI-compatible OS is shown in Example 11-25:

Example 11-25. A Typical OS Idle Loop

```

// WorkQueue is a memory location indicating there is a thread
// ready to run. A non-zero value for WorkQueue is assumed to
// indicate the presence of work to be scheduled on the processor.
// The idle loop is entered with interrupts disabled.

WHILE (1) {
    IF (WorkQueue) THEN {
        // Schedule work at WorkQueue.
    }
    ELSE {
        // No work to do - wait in appropriate C-state handler depending
        // on Idle time accumulated
        IF (IdleTime >= IdleTimeThreshold) THEN {
            // Call appropriate C1, C2, C3 state handler, C1 handler
            // shown below
        }
    }
}
// C1 handler uses a Halt instruction
VOID C1Handler()
{ STI
  HLT
}

```

The MONITOR and MWAIT instructions may be considered for use in the C0 idle state loops, if MONITOR and MWAIT are supported.

Example 11-26. An OS Idle Loop with MONITOR/MWAIT in the C0 Idle Loop

```

// WorkQueue is a memory location indicating there is a thread
// ready to run. A non-zero value for WorkQueue is assumed to
// indicate the presence of work to be scheduled on the processor.
// The following example assumes that the necessary padding has been
// added surrounding WorkQueue to eliminate false wakeups
// The idle loop is entered with interrupts disabled.

WHILE (1) {
    IF (WorkQueue) THEN {
        // Schedule work at WorkQueue.
    }
    ELSE {
        // No work to do - wait in appropriate C-state handler depending
        // on Idle time accumulated.
        IF (IdleTime >= IdleTimeThreshold) THEN {
            // Call appropriate C1, C2, C3 state handler, C1
            // handler shown below
            MONITOR WorkQueue // Setup of eax with WorkQueue
                                // LinearAddress,
                                // ECX, EDX = 0
            IF (WorkQueue = 0) THEN {
                MWAIT
            }
        }
    }
}

```

```

}
// C1 handler uses a Halt instruction.
VOID C1Handler()
{   STI
    HLT
}

```

11.10.6.3 Halt Idle Logical Processors

If one of two logical processors is idle or in a spin-wait loop of long duration, explicitly halt that processor by means of a HLT instruction.

In an MP system, operating systems can place idle processors into a loop that continuously checks the run queue for runnable software tasks. Logical processors that execute idle loops consume a significant amount of core's execution resources that might otherwise be used by the other logical processors in the physical package. For this reason, halting idle logical processors optimizes the performance.¹ If all logical processors within a physical package are halted, the processor will enter a power-saving state.

11.10.6.4 Potential Usage of MONITOR/MWAIT in C1 Idle Loops

An operating system may also consider replacing HLT with MONITOR/MWAIT in its C1 idle loop. An example is shown in Example 11-27:

Example 11-27. An OS Idle Loop with MONITOR/MWAIT in the C1 Idle Loop

```

// WorkQueue is a memory location indicating there is a thread
// ready to run. A non-zero value for WorkQueue is assumed to
// indicate the presence of work to be scheduled on the processor.
// The following example assumes that the necessary padding has been
// added surrounding WorkQueue to eliminate false wakeups
// The idle loop is entered with interrupts disabled.

```

```

WHILE (1) {
    IF (WorkQueue) THEN {
        // Schedule work at WorkQueue
    }
    ELSE {
        // No work to do - wait in appropriate C-state handler depending
        // on Idle time accumulated
        IF (IdleTime >= IdleTimeThreshold) THEN {
            // Call appropriate C1, C2, C3 state handler, C1
            // handler shown below
        }
    }
}

```

```

VOID C1Handler()

{   MONITOR WorkQueue // Setup of eax with WorkQueue LinearAddress,
                        // ECX, EDX = 0
    IF (WorkQueue = 0) THEN {
        STI
    }
}

```

1. Excessive transitions into and out of the HALT state could also incur performance penalties. Operating systems should evaluate the performance trade-offs for their operating system.

```

    MWAIT    // EAX, ECX = 0
  }
}

```

11.10.6.5 Guidelines for Scheduling Threads on Logical Processors Sharing Execution Resources

Because the logical processors, the order in which threads are dispatched to logical processors for execution can affect the overall efficiency of a system. The following guidelines are recommended for scheduling threads for execution.

- Dispatch threads to one logical processor per processor core before dispatching threads to the other logical processor sharing execution resources in the same processor core.
- In an MP system with two or more physical packages, distribute threads out over all the physical processors, rather than concentrate them in one or two physical processors.
- Use processor affinity to assign a thread to a specific processor core or package, depending on the cache-sharing topology. The practice increases the chance that the processor's caches will contain some of the thread's code and data when it is dispatched for execution after being suspended.

11.10.6.6 Eliminate Execution-Based Timing Loops

Intel discourages the use of timing loops that depend on a processor's execution speed to measure time. There are several reasons:

- Timing loops cause problems when they are calibrated on a IA-32 processor running at one frequency and then executed on a processor running at another frequency.
- Routines for calibrating execution-based timing loops produce unpredictable results when run on an IA-32 processor supporting Intel Hyper-Threading Technology. This is due to the sharing of execution resources between the logical processors within a physical package.

To avoid the problems described, timing loop routines must use a timing mechanism for the loop that does not depend on the execution speed of the logical processors in the system. The following sources are generally available:

- A high resolution system timer (for example, an Intel 8254).
- A high resolution timer within the processor (such as, the local APIC timer or the time-stamp counter).

For additional information, see the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

11.10.6.7 Place Locks and Semaphores in Aligned, 128-Byte Blocks of Memory

When software uses locks or semaphores to synchronize processes, threads, or other code sections; Intel recommends that only one lock or semaphore be present within a cache line (or 128 byte sector, if 128-byte sector is supported). In processors based on Intel NetBurst microarchitecture (which support 128-byte sector consisting of two cache lines), following this recommendation means that each lock or semaphore should be contained in a 128-byte block of memory that begins on a 128-byte boundary. The practice minimizes the bus traffic required to service locks.

11.11 MP INITIALIZATION FOR P6 FAMILY PROCESSORS

This section describes the MP initialization process for systems that use multiple P6 family processors. This process uses the MP initialization protocol that was introduced with the Pentium Pro processor (see Section 11.4, "Multiple-Processor (MP) Initialization"). For P6 family processors, this protocol is typically used to boot 2 or 4 processors that reside on single system bus; however, it can support from 2 to 15 processors in a multi-clustered system when the APIC buses are tied together. Larger systems are not supported.

11.11.1 Overview of the MP Initialization Process for P6 Family Processors

During the execution of the MP initialization protocol, one processor is selected as the bootstrap processor (BSP) and the remaining processors are designated as application processors (APs), see Section 11.4.1, “BSP and AP Processors.” Thereafter, the BSP manages the initialization of itself and the APs. This initialization includes executing BIOS initialization code and operating-system initialization code.

The MP protocol imposes the following requirements and restrictions on the system:

- An APIC clock (APICLK) must be provided.
- The MP protocol will be executed only after a power-up or RESET. If the MP protocol has been completed and a BSP has been chosen, subsequent INITs (either to a specific processor or system wide) do not cause the MP protocol to be repeated. Instead, each processor examines its BSP flag (in the APIC_BASE MSR) to determine whether it should execute the BIOS boot-strap code (if it is the BSP) or enter a wait-for-SIPI state (if it is an AP).
- All devices in the system that are capable of delivering interrupts to the processors must be inhibited from doing so for the duration of the MP initialization protocol. The time during which interrupts must be inhibited includes the window between when the BSP issues an INIT-SIPI-SIPI sequence to an AP and when the AP responds to the last SIPI in the sequence.

The following special-purpose interprocessor interrupts (IPIs) are used during the boot phase of the MP initialization protocol. These IPIs are broadcast on the APIC bus.

- Boot IPI (BIPI)—Initiates the arbitration mechanism that selects a BSP from the group of processors on the system bus and designates the remainder of the processors as APs. Each processor on the system bus broadcasts a BIPI to all the processors following a power-up or RESET.
- Final Boot IPI (FIPI)—Initiates the BIOS initialization procedure for the BSP. This IPI is broadcast to all the processors on the system bus, but only the BSP responds to it. The BSP responds by beginning execution of the BIOS initialization code at the reset vector.
- Startup IPI (SIPI)—Initiates the initialization procedure for an AP. The SIPI message contains a vector to the AP initialization code in the BIOS.

Table 11-5 describes the various fields of the boot phase IPIs.

Table 11-5. Boot Phase IPI Message Format

Type	Destination Field	Destination Shorthand	Trigger Mode	Level	Destination Mode	Delivery Mode	Vector (Hex)
BIPI	Not used	All including self	Edge	Deassert	Don't Care	Fixed (000)	40 to 4E*
FIPI	Not used	All including self	Edge	Deassert	Don't Care	Fixed (000)	10
SIPI	Used	All excluding self	Edge	Assert	Physical	StartUp (110)	00 to FF

NOTE:

* For all P6 family processors.

For BIPI messages, the lower 4 bits of the vector field contain the APIC ID of the processor issuing the message and the upper 4 bits contain the “generation ID” of the message. All P6 family processor will have a generation ID of 4H. BIPIs will therefore use vector values ranging from 40H to 4EH (4FH can not be used because FH is not a valid APIC ID).

11.11.2 MP Initialization Protocol Algorithm

Following a power-up or RESET of a system, the P6 family processors in the system execute the MP initialization protocol algorithm to initialize each of the processors on the system bus. In the course of executing this algorithm, the following boot-up and initialization operations are carried out:

1. Each processor on the system bus is assigned a unique APIC ID, based on system topology (see Section 11.4.5, "Identifying Logical Processors in an MP System"). This ID is written into the local APIC ID register for each processor.
2. Each processor executes its internal BIST simultaneously with the other processors on the system bus. Upon completion of the BIST (at T₀), each processor broadcasts a BIPI to "all including self" (see Figure 11-8).
3. APIC arbitration hardware causes all the APICs to respond to the BIPIs one at a time (at T₁, T₂, T₃, and T₄).
4. When the first BIPI is received (at time T₁), each APIC compares the four least significant bits of the BIPI's vector field with its APIC ID. If the vector and APIC ID match, the processor selects itself as the BSP by setting the BSP flag in its IA32_APIC_BASE MSR. If the vector and APIC ID do not match, the processor selects itself as an AP by entering the "wait for SIPI" state. (Note that in Figure 11-8, the BIPI from processor 1 is the first BIPI to be handled, so processor 1 becomes the BSP.)
5. The newly established BSP broadcasts an FIPI message to "all including self." The FIPI is guaranteed to be handled only after the completion of the BIPIs that were issued by the non-BSP processors.

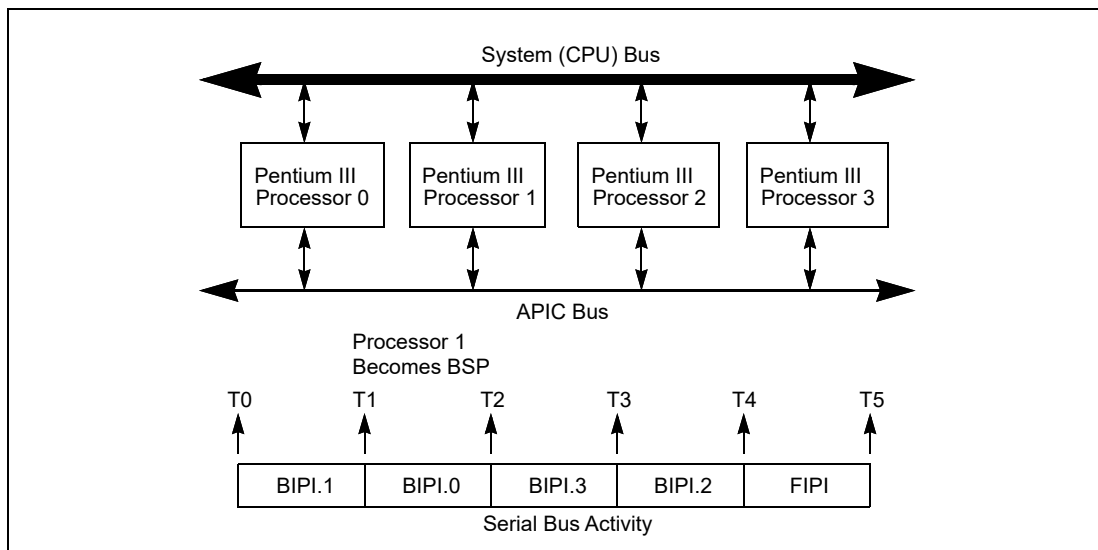


Figure 11-8. MP System With Multiple Pentium III Processors

6. After the BSP has been established, the outstanding BIPIs are received one at a time (at T₂, T₃, and T₄) and ignored by all processors.
7. When the FIPI is finally received (at T₅), only the BSP responds to it. It responds by fetching and executing BIOS boot-strap code, beginning at the reset vector (physical address FFFF FFF0H).
8. As part of the boot-strap code, the BSP creates an ACPI table and an MP table and adds its initial APIC ID to these tables as appropriate.
9. At the end of the boot-strap procedure, the BSP broadcasts a SIPI message to all the APs in the system. Here, the SIPI message contains a vector to the BIOS AP initialization code (at 000V V000H, where VV is the vector contained in the SIPI message).
10. All APs respond to the SIPI message by racing to a BIOS initialization semaphore. The first one to the semaphore begins executing the initialization code. (See MP init code for semaphore implementation details.) As part of the AP initialization procedure, the AP adds its APIC ID number to the ACPI and MP tables as appropriate. At the completion of the initialization procedure, the AP executes a CLI instruction (to clear the IF flag in the EFLAGS register) and halts itself.
11. When each of the APs has gained access to the semaphore and executed the AP initialization code and all written their APIC IDs into the appropriate places in the ACPI and MP tables, the BSP establishes a count for the number of processors connected to the system bus, completes executing the BIOS boot-strap code, and then begins executing operating-system boot-strap and start-up code.

12. While the BSP is executing operating-system boot-strap and start-up code, the APs remain in the halted state. In this state they will respond only to INITs, NMIs, and SMIs. They will also respond to snoops and to assertions of the STPCLK# pin.

See Section 11.4.4, “MP Initialization Example,” for an annotated example the use of the MP protocol to boot IA-32 processors in an MP. This code should run on any IA-32 processor that used the MP protocol.

11.11.2.1 Error Detection and Handling During the MP Initialization Protocol

Errors may occur on the APIC bus during the MP initialization phase. These errors may be transient or permanent and can be caused by a variety of failure mechanisms (for example, broken traces, soft errors during bus usage, etc.). All serial bus related errors will result in an APIC checksum or acceptance error.

The MP initialization protocol makes the following assumptions regarding errors that occur during initialization:

- If errors are detected on the APIC bus during execution of the MP initialization protocol, the processors that detect the errors are shut down.
- The MP initialization protocol will be executed by processors even if they fail their BIST sequences.

17. Updates to Chapter 12, Volume 3A

Change bars and text show changes to Chapter 12 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- In Section 12.11.8, "Optional Processor Microcode Update Specifications," specified that authentication of the microcode update is BIOS responsibility before writing it to flash.

CHAPTER 12

PROCESSOR MANAGEMENT AND INITIALIZATION

This chapter describes the facilities provided for managing processor wide functions and for initializing the processor. The subjects covered include: processor initialization, x87 FPU initialization, processor configuration, feature determination, mode switching, the MSRs (in the Pentium, P6 family, Pentium 4, and Intel Xeon processors), and the MTRRs (in the P6 family, Pentium 4, and Intel Xeon processors).

12.1 INITIALIZATION OVERVIEW

Following power-up or an assertion of the RESET# pin, each processor on the system bus performs a hardware initialization of the processor (known as a hardware reset) and an optional built-in self-test (BIST). A hardware reset sets each processor's registers to a known state and places the processor in real-address mode. It also invalidates the internal caches, translation lookaside buffers (TLBs) and the branch target buffer (BTB). At this point, the action taken depends on the processor family:

- **Pentium 4 processors (CUID DisplayFamily 0FH)** — All the processors on the system bus (including a single processor in a uniprocessor system) execute the multiple processor (MP) initialization protocol. The processor that is selected through this protocol as the bootstrap processor (BSP) then immediately starts executing software-initialization code in the current code segment beginning at the offset in the EIP register. The application (non-BSP) processors (APs) go into a Wait For Startup IPI (SIPI) state while the BSP is executing initialization code. See Section 11.4, "Multiple-Processor (MP) Initialization," for more details. Note that in a uniprocessor system, the single Pentium 4 or Intel Xeon processor automatically becomes the BSP.
- **IA-32 and Intel 64 processors (CUID DisplayFamily 06H)** — The action taken is the same as for the Pentium 4 processors (as described in the previous paragraph).
- **Pentium processors** — In either a single- or dual- processor system, a single Pentium processor is always pre-designated as the primary processor. Following a reset, the primary processor behaves as follows in both single- and dual-processor systems. Using the dual-processor (DP) ready initialization protocol, the primary processor immediately starts executing software-initialization code in the current code segment beginning at the offset in the EIP register. The secondary processor (if there is one) goes into a halt state.
- **Intel486 processor** — The primary processor (or single processor in a uniprocessor system) immediately starts executing software-initialization code in the current code segment beginning at the offset in the EIP register. (The Intel486 does not automatically execute a DP or MP initialization protocol to determine which processor is the primary processor.)

The software-initialization code performs all system-specific initialization of the BSP or primary processor and the system logic.

At this point, for MP (or DP) systems, the BSP (or primary) processor wakes up each AP (or secondary) processor to enable those processors to execute self-configuration code.

When all processors are initialized, configured, and synchronized, the BSP or primary processor begins executing an initial operating-system or executive task.

The x87 FPU is also initialized to a known state during hardware reset. x87 FPU software initialization code can then be executed to perform operations such as setting the precision of the x87 FPU and the exception masks. No special initialization of the x87 FPU is required to switch operating modes.

Asserting the INIT# pin on the processor invokes a similar response to a hardware reset. The major difference is that during an INIT, the internal caches, MSRs, MTRRs, and x87 FPU state are left unchanged (although, the TLBs and BTB are invalidated as with a hardware reset). An INIT provides a method for switching from protected to real-address mode while maintaining the contents of the internal caches.

12.1.1 Processor State After Reset

Following power-up, The state of control register CR0 is 60000010H (see Figure 12-1). This places the processor is in real-address mode with paging disabled.

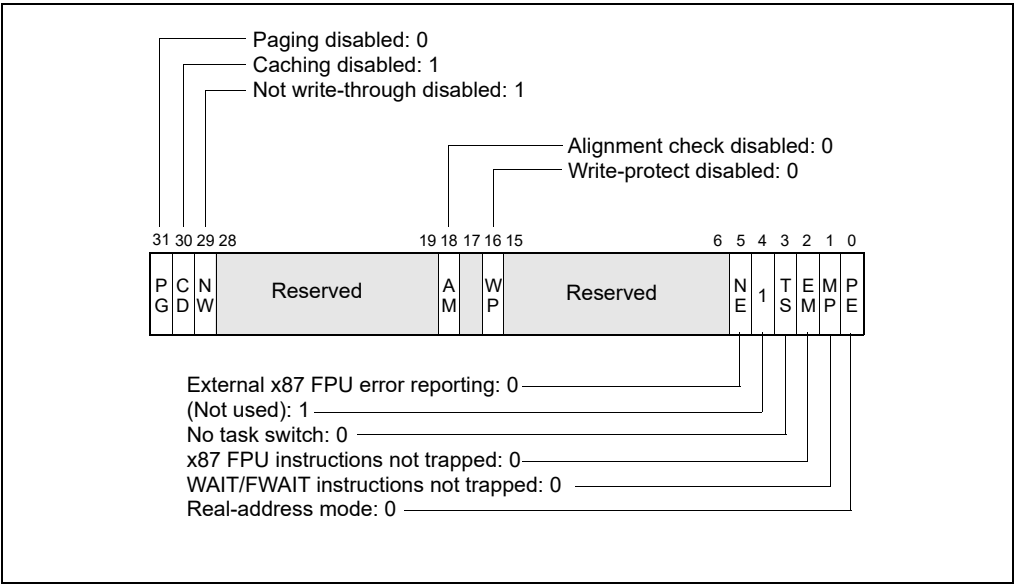


Figure 12-1. Contents of CR0 Register after Reset

The state of the flags and other registers following power-up for the Pentium 4, Pentium Pro, and Pentium processors are shown in Section 25.39, “Initial State of Pentium, Pentium Pro and Pentium 4 Processors,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B.

Table 12-1 shows processor states of IA-32 and Intel 64 processors with CPUID DisplayFamily signature of 06H at the following events: power-up, RESET, and INIT. In a few cases, the behavior of some registers behave slightly different across warm RESET, the variant cases are marked in Table 12-1 and described in more detail in Table 12-2.

Table 12-1. IA-32 and Intel® 64 Processor States Following Power-up, Reset, or INIT

Register	Power up	Reset	INIT
EFLAGS ¹	00000002H	00000002H	00000002H
EIP	0000FFF0H	0000FFF0H	0000FFF0H
CR0	60000010H ²	60000010H ²	60000010H ²
CR2, CR3, CR4	00000000H	00000000H	00000000H
CS	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed
SS, DS, ES, FS, GS	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed
EDX	000n06xxH ³	000n06xxH ³	000n06xxH ³
EAX	0 ⁴	0 ⁴	0 ⁴
EBX, ECX, ESI, EDI, EBP, ESP	00000000H	00000000H	00000000H

Table 12-1. IA-32 and Intel® 64 Processor States Following Power-up, Reset, or INIT (Contd.)

Register	Power up	Reset	INIT
ST0 through ST7 ⁵	+0.0	+0.0	FINIT/FNINIT: Unchanged
x87 FPU Control Word ⁵	0040H	0040H	FINIT/FNINIT: 037FH
x87 FPU Status Word ⁵	0000H	0000H	FINIT/FNINIT: 0000H
x87 FPU Tag Word ⁵	5555H	5555H	FINIT/FNINIT: FFFFH
x87 FPU Data Operand and CS Seg. Selectors ⁵	0000H	0000H	FINIT/FNINIT: 0000H
x87 FPU Data Operand and Inst. Pointers ⁵	00000000H	00000000H	FINIT/FNINIT: 00000000H
MM0 through MM7 ⁵	0000000000000000H	0000000000000000H	INIT or FINIT/FNINIT: Unchanged
XMM0 through XMM7	0H	0H	Unchanged
MXCSR	1F80H	1F80H	Unchanged
GDTR, IDTR	Base = 00000000H Limit = FFFFH AR = Present, R/W	Base = 00000000H Limit = FFFFH AR = Present, R/W	Base = 00000000H Limit = FFFFH AR = Present, R/W
LDTR, Task Register	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W
DR0, DR1, DR2, DR3	00000000H	00000000H	00000000H
DR6	FFFF0FF0H	FFFF0FF0H	FFFF0FF0H
DR7	00000400H	00000400H	00000400H
R8-R15	0000000000000000H	0000000000000000H	0000000000000000H
XMM8-XMM15	0H	0H	Unchanged
XCR0	1H	1H	Unchanged
IA32_XSS	0H	0H	Unchanged
YMM_H[255:128]	0H	0H	Unchanged
BNDCFGU	0H	0H	0H
BND0-BND3	0H	0H	0H
IA32_BNDCFGS	0H	0H	0H
OPMASK	0H	0H	Unchanged
ZMM_H[511:256]	0H	0H	Unchanged
ZMMHi16[511:0]	0H	0H	Unchanged
PKRU	0H	0H	Unchanged
Intel Processor Trace MSRs	0H	0H ^w	Unchanged
Time-Stamp Counter	0H	0H ^w	Unchanged
IA32_TSC_AUX	0H	0H	Unchanged
IA32_TSC_ADJUST	0H	0H	Unchanged
IA32_TSC_DEADLINE	0H	0H	Unchanged
IA32_SYSENTER_CS/ESP/EIP	0H	0H	Unchanged
IA32_EFER	0000000000000000H	0000000000000000H	0000000000000000H
IA32_STAR/LSTAR	0H	0H	Unchanged

Table 12-1. IA-32 and Intel® 64 Processor States Following Power-up, Reset, or INIT (Contd.)

Register	Power up	Reset	INIT
IA32_FS_BASE/GS_BASE	0H	0H	0H
IA32_PMCx, IA32_PERFEVTSELx	0H	0H	Unchanged
IA32_PERF_GLOBAL_CTRL	Sets bits n-1:0 and clears the upper bits. ⁷	Sets bits n-1:0 and clears the upper bits. ⁷	Unchanged
IA32_FIXED_CTRx, IA32_FIXED_CTR_CTRL	0H	0H	Unchanged
Data and Code Cache, TLBs	Invalid ⁶	Invalid ⁶	Unchanged
Fixed MTRRs	Disabled	Disabled	Unchanged
Variable MTRRs	Disabled	Disabled	Unchanged
Machine-Check Banks	Undefined	Undefined ^w	Unchanged
Last Branch Record Stack	0	0 ^w	Unchanged
APIC	Enabled	Enabled	Unchanged
X2APIC	Disabled	Disabled	Unchanged
IA32_DEBUG_INTERFACE	0	0 ^w	Unchanged

NOTES:

1. The 10 most-significant bits of the EFLAGS register are undefined following a reset. Software should not depend on the states of any of these bits.
2. The CD and NW flags are unchanged, bit 4 is set to 1, all other bits are cleared.
3. Where “n” is the Extended Model Value for the respective processor, and “xx” = don't care.
4. If Built-In Self-Test (BIST) is invoked on power up or reset, EAX is 0 only if all tests passed. (BIST cannot be invoked during an INIT.)
5. The state of the x87 FPU and MMX registers is not changed by the execution of an INIT.
6. Internal caches are invalid after power-up and RESET, but left unchanged with an INIT.
7. Where “n” is the number of general-purpose counters available in the processor. See Chapter 22, “Performance Monitoring,” for additional details.

^w: Warm RESET behavior differs from power-on RESET with details listed in Table 12-2.

Table 12-2. Variance of RESET Values in Selected Intel Architecture Processors

State	XREF	Value	Feature Flag or DisplayFamily_DisplayModel Signatures
Time-Stamp Counter	Warm RESET	Unmodified across warm Reset	06_2DH, 06_3EH
Machine-Check Banks	Warm RESET	IA32_MCi_Status banks are unmodified across warm Reset	06_2DH, 06_3EH, 06_3FH, 06_4FH, 06_56H
Last Branch Record Stack	Warm RESET	LBR stack MSRs are unmodified across warm Reset	06_1AH, 06_1CH, DisplayFamiy= 06 and DisplayModel >1DH
Intel Processor Trace MSRs	Warm RESET	Clears IA32_RTIT_CTL.TraceEn, the rest of MSRs are unmodified	If CPUID.14H.00H:EBX[2] = 1
IA32_DEBUG_INTERFACE	Warm RESET	Unmodified across warm Reset	If CPUID.01H:ECX[11] = 1

12.1.2 Processor Built-In Self-Test (BIST)

Hardware may request that the BIST be performed at power-up. The EAX register is cleared (0H) if the processor passes the BIST. A nonzero value in the EAX register after the BIST indicates that a processor fault was detected. If the BIST is not requested, the contents of the EAX register after a hardware reset is 0H.

The overhead for performing a BIST varies between processor families. For example, the BIST takes approximately 30 million processor clock periods to execute on the Pentium 4 processor. This clock count is model-specific; Intel reserves the right to change the number of periods for any Intel 64 or IA-32 processor, without notification.

12.1.3 Model and Stepping Information

Following a hardware reset, the EDX register contains component identification and revision information (see Figure 12-2). For example, the model, family, and processor type returned for the first processor in the Intel Pentium 4 family is as follows: model (0000B), family (1111B), and processor type (00B).

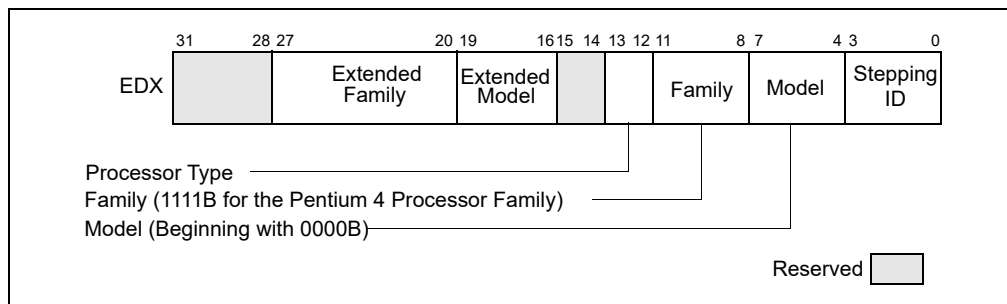


Figure 12-2. Version Information in the EDX Register after Reset

The stepping ID field contains a unique identifier for the processor's stepping ID or revision level. The extended family and extended model fields were added to the IA-32 architecture in the Pentium 4 processors.

12.1.4 First Instruction Executed

The first instruction that is fetched and executed following a hardware reset is located at physical address FFFFFFF0H. This address is 16 bytes below the processor's uppermost physical address. The EPROM containing the software-initialization code must be located at this address.

The address FFFFFFF0H is beyond the 1-MByte addressable range of the processor while in real-address mode. The processor is initialized to this starting address as follows. The CS register has two parts: the visible segment selector part and the hidden base address part. In real-address mode, the base address is normally formed by shifting the 16-bit segment selector value 4 bits to the left to produce a 20-bit base address. However, during a hardware reset, the segment selector in the CS register is loaded with F000H and the base address is loaded with FFFF0000H. The starting address is thus formed by adding the base address to the value in the EIP register (that is, FFFF0000 + FFF0H = FFFFFFF0H).

The first time the CS register is loaded with a new value after a hardware reset, the processor will follow the normal rule for address translation in real-address mode (that is, [CS base address = CS segment selector * 16]). To ensure that the base address in the CS register remains unchanged until the EPROM based software-initialization code is completed, the code must not contain a far jump or far call or allow an interrupt to occur (which would cause the CS selector value to be changed).

12.2 X87 FPU INITIALIZATION

Software-initialization code can determine whether the processor contains an x87 FPU by using the CPUID instruction. The code must then initialize the x87 FPU and set flags in control register CR0 to reflect the state of the x87 FPU environment.

A hardware reset places the x87 FPU in the state shown in Table 12-1. This state is different from the state the x87 FPU is placed in following the execution of an FINIT or FNINIT instruction (also shown in Table 12-1). If the x87 FPU is to be used, the software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. These instructions, tag all data registers as empty, clear all the exception masks, set the TOP-of-stack value to 0, and select the default rounding and precision controls setting (round to nearest and 64-bit precision).

If the processor is reset by asserting the INIT# pin, the x87 FPU state is not changed.

12.2.1 Configuring the x87 FPU Environment

Initialization code must load the appropriate values into the MP, EM, and NE flags of control register CR0. These bits are cleared on hardware reset of the processor. Figure 12-3 shows the suggested settings for these flags, depending on the IA-32 processor being initialized. Initialization code can test for the type of processor present before setting or clearing these flags.

Table 12-3. Recommended Settings of EM and MP Flags on IA-32 Processors

EM	MP	NE	IA-32 processor
1	0	1	Intel486™ SX, Intel386™ DX, and Intel386™ SX processors only, without the presence of a math coprocessor.
0	1	1 or 0*	Pentium 4, Intel Xeon, P6 family, Pentium, Intel486™ DX, and Intel 487 SX processors, and Intel386 DX and Intel386 SX processors when a companion math coprocessor is present.
0	1	1 or 0*	More recent Intel 64 or IA-32 processors

NOTE:
* The setting of the NE flag depends on the operating system being used.

The EM flag determines whether floating-point instructions are executed by the x87 FPU (EM is cleared) or a device-not-available exception (#NM) is generated for all floating-point instructions so that an exception handler can emulate the floating-point operation (EM = 1). Ordinarily, the EM flag is cleared when an x87 FPU or math coprocessor is present and set if they are not present. If the EM flag is set and no x87 FPU, math coprocessor, or floating-point emulator is present, the processor will hang when a floating-point instruction is executed.

The MP flag determines whether WAIT/FWAIT instructions react to the setting of the TS flag. If the MP flag is clear, WAIT/FWAIT instructions ignore the setting of the TS flag; if the MP flag is set, they will generate a device-not-available exception (#NM) if the TS flag is set. Generally, the MP flag should be set for processors with an integrated x87 FPU and clear for processors without an integrated x87 FPU and without a math coprocessor present. However, an operating system can choose to save the floating-point context at every context switch, in which case there would be no need to set the MP bit.

Table 2-2 shows the actions taken for floating-point and WAIT/FWAIT instructions based on the settings of the EM, MP, and TS flags.

The NE flag determines whether unmasked floating-point exceptions are handled by generating a floating-point error exception internally (NE is set, native mode) or through an external interrupt (NE is cleared). In systems where an external interrupt controller is used to invoke numeric exception handlers (such as MS-DOS-based systems), the NE bit should be cleared.

12.2.2 Setting the Processor for x87 FPU Software Emulation

Setting the EM flag causes the processor to generate a device-not-available exception (#NM) and trap to a software exception handler whenever it encounters a floating-point instruction. (Table 12-3 shows when it is appropriate to use this flag.) Setting this flag has two functions:

- It allows x87 FPU code to run on an IA-32 processor that has neither an integrated x87 FPU nor is connected to an external math coprocessor, by using a floating-point emulator.
- It allows floating-point code to be executed using a special or nonstandard floating-point emulator, selected for a particular application, regardless of whether an x87 FPU or math coprocessor is present.

To emulate floating-point instructions, the EM, MP, and NE flag in control register CR0 should be set as shown in Table 12-4.

Table 12-4. Software Emulation Settings of EM, MP, and NE Flags

CRO Bit	Value
EM	1
MP	0
NE	1

Regardless of the value of the EM bit, the Intel486 SX processor generates a device-not-available exception (#NM) upon encountering any floating-point instruction.

12.3 CACHE ENABLING

IA-32 processors (beginning with the Intel486 processor) and Intel 64 processors contain internal instruction and data caches. These caches are enabled by clearing the CD and NW flags in control register CR0. (They are set during a hardware reset.) Because all internal cache lines are invalid following reset initialization, it is not necessary to invalidate the cache before enabling caching. Any external caches may require initialization and invalidation using a system-specific initialization and invalidation code sequence.

Depending on the hardware and operating system or executive requirements, additional configuration of the processor's caching facilities will probably be required. Beginning with the Intel486 processor, page-level caching can be controlled with the PCD and PWT flags in page-directory and page-table entries. Beginning with the P6 family processors, the memory type range registers (MTRRs) control the caching characteristics of the regions of physical memory. (For the Intel486 and Pentium processors, external hardware can be used to control the caching characteristics of regions of physical memory.) See Chapter 14, "Memory Cache Control," for detailed information on configuration of the caching facilities in the Pentium 4, Intel Xeon, and P6 family processors and system memory.

12.4 MODEL-SPECIFIC REGISTERS (MSRS)

Most IA-32 processors (starting from Pentium processors) and Intel 64 processors contain a model-specific registers (MSRs). A given MSR may not be supported across all families and models for Intel 64 and IA-32 processors. Some MSRs are designated as architectural to simplify software programming; a feature introduced by an architectural MSR is expected to be supported in future processors. Non-architectural MSRs are not guaranteed to be supported or to have the same functions on future processors.

MSRs that provide control for a number of hardware and software-related features, include:

- Performance-monitoring counters (see Chapter 22, "Performance Monitoring").
- Debug extensions (see Chapter 20, "Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features").
- Machine-check exception capability and its accompanying machine-check architecture (see Chapter 18, "Machine-Check Architecture").
- MTRRs (see Section 14.11, "Memory Type Range Registers (MTRRs)").
- Thermal and power management.
- Instruction-specific support (for example: SYSENTER, SYSEXIT, SWAPGS, etc.).
- Processor feature/mode support (for example: IA32_EFER, IA32_FEATURE_CONTROL).

The MSRs can be read and written to using the RDMSR and WRMSR instructions, respectively.

When performing software initialization of an IA-32 or Intel 64 processor, many of the MSRs will need to be initialized to set up things like performance-monitoring events, run-time machine checks, and memory types for physical memory.

Lists of available performance-monitoring events can be found at: <https://perfmon-events.intel.com/>, and lists of available MSRs are given in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4. The references earlier in this section show where the functions of the various groups of MSRs are described in this manual.

12.5 MEMORY TYPE RANGE REGISTERS (MTRRS)

Memory type range registers (MTRRs) were introduced into the IA-32 architecture with the Pentium Pro processor. They allow the type of caching (or no caching) to be specified in system memory for selected physical address ranges. They allow memory accesses to be optimized for various types of memory such as RAM, ROM, frame buffer memory, and memory-mapped I/O devices.

In general, initializing the MTRRs is normally handled by the software initialization code or BIOS and is not an operating system or executive function. At the very least, all the MTRRs must be cleared to 0, which selects the uncached (UC) memory type. See Section 14.11, “Memory Type Range Registers (MTRRs),” for detailed information on the MTRRs.

12.6 INITIALIZING SSE/SSE2/SSE3/SSSE3 EXTENSIONS

For processors that contain SSE/SSE2/SSE3/SSSE3 extensions, steps must be taken when initializing the processor to allow execution of these instructions.

1. Check the CPUID feature flags for the presence of the SSE/SSE2/SSE3/SSSE3 extensions (respectively: EDX bits 25 and 26, ECX bit 0 and 9) and support for the FXSAVE and FXRSTOR instructions (EDX bit 24). Also check for support for the CLFLUSH instruction (EDX bit 19). The CPUID feature flags are loaded in the EDX and ECX registers when the CPUID instruction is executed with a 1 in the EAX register.
2. Set the OSFXSR flag (bit 9 in control register CR4) to indicate that the operating system supports saving and restoring the SSE/SSE2/SSE3/SSSE3 execution environment (XMM and MXCSR registers) with the FXSAVE and FXRSTOR instructions, respectively. See Section 2.5, “Control Registers,” for a description of the OSFXSR flag.
3. Set the OSXMMEXCPT flag (bit 10 in control register CR4) to indicate that the operating system supports the handling of SSE/SSE2/SSE3 SIMD floating-point exceptions (#XM). See Section 2.5, “Control Registers,” for a description of the OSXMMEXCPT flag.
4. Set the mask bits and flags in the MXCSR register according to the mode of operation desired for SSE/SSE2/SSE3 SIMD floating-point instructions. See “MXCSR Control and Status Register” in Chapter 10, “Programming with Intel® Streaming SIMD Extensions (Intel® SSE),” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for a detailed description of the bits and flags in the MXCSR register.

12.7 SOFTWARE INITIALIZATION FOR REAL-ADDRESS MODE OPERATION

Following a hardware reset (either through a power-up or the assertion of the RESET# pin) the processor is placed in real-address mode and begins executing software initialization code from physical address FFFFFFF0H. Software initialization code must first set up the necessary data structures for handling basic system functions, such as a real-mode IDT for handling interrupts and exceptions. If the processor is to remain in real-address mode, software must then load additional operating-system or executive code modules and data structures to allow reliable execution of application programs in real-address mode.

If the processor is going to operate in protected mode, software must load the necessary data structures to operate in protected mode and then switch to protected mode. The protected-mode data structures that must be loaded are described in Section 12.8, “Software Initialization for Protected-Mode Operation.”

12.7.1 Real-Address Mode IDT

In real-address mode, the only system data structure that must be loaded into memory is the IDT (also called the “interrupt vector table”). By default, the address of the base of the IDT is physical address 0H. This address can be

changed by using the LIDT instruction to change the base address value in the IDTR. Software initialization code needs to load interrupt- and exception-handler pointers into the IDT before interrupts can be enabled.

The actual interrupt- and exception-handler code can be contained either in EPROM or RAM; however, the code must be located within the 1-MByte addressable range of the processor in real-address mode. If the handler code is to be stored in RAM, it must be loaded along with the IDT.

12.7.2 NMI Interrupt Handling

The NMI interrupt is always enabled (except when multiple NMIs are nested). If the IDT and the NMI interrupt handler need to be loaded into RAM, there will be a period of time following hardware reset when an NMI interrupt cannot be handled. During this time, hardware must provide a mechanism to prevent an NMI interrupt from halting code execution until the IDT and the necessary NMI handler software is loaded. Here are two examples of how NMIs can be handled during the initial states of processor initialization:

- A simple IDT and NMI interrupt handler can be provided in EPROM. This allows an NMI interrupt to be handled immediately after reset initialization.
- The system hardware can provide a mechanism to enable and disable NMIs by passing the NMI# signal through an AND gate controlled by a flag in an I/O port. Hardware can clear the flag when the processor is reset, and software can set the flag when it is ready to handle NMI interrupts.

12.8 SOFTWARE INITIALIZATION FOR PROTECTED-MODE OPERATION

The processor is placed in real-address mode following a hardware reset. At this point in the initialization process, some basic data structures and code modules must be loaded into physical memory to support further initialization of the processor, as described in Section 12.7, “Software Initialization for Real-Address Mode Operation.” Before the processor can be switched to protected mode, the software initialization code must load a minimum number of protected mode data structures and code modules into memory to support reliable operation of the processor in protected mode. These data structures include the following:

- A IDT.
- A GDT.
- A TSS.
- (Optional) An LDT.
- If paging is to be used, at least one page directory and one page table.
- A code segment that contains the code to be executed when the processor switches to protected mode.
- One or more code modules that contain the necessary interrupt and exception handlers.

Software initialization code must also initialize the following system registers before the processor can be switched to protected mode:

- The GDTR.
- (Optional.) The IDTR. This register can also be initialized immediately after switching to protected mode, prior to enabling interrupts.
- Control registers CR1 through CR4.
- (Pentium 4, Intel Xeon, and P6 family processors only.) The memory type range registers (MTRRs).

With these data structures, code modules, and system registers initialized, the processor can be switched to protected mode by loading control register CR0 with a value that sets the PE flag (bit 0).

12.8.1 Protected-Mode System Data Structures

The contents of the protected-mode system data structures loaded into memory during software initialization, depend largely on the type of memory management the protected-mode operating-system or executive is going to support: flat, flat with paging, segmented, or segmented with paging.

To implement a flat memory model without paging, software initialization code must at a minimum load a GDT with one code and one data-segment descriptor. A null descriptor in the first GDT entry is also required. The stack can be placed in a normal read/write data segment, so no dedicated descriptor for the stack is required. A flat memory model with paging also requires a page directory and at least one page table (unless all pages are 4 MBytes in which case only a page directory is required). See Section 12.8.3, “Initializing Paging.”

Before the GDT can be used, the base address and limit for the GDT must be loaded into the GDTR register using an LGDT instruction.

A multi-segmented model may require additional segments for the operating system, as well as segments and LDTs for each application program. LDTs require segment descriptors in the GDT. Some operating systems allocate new segments and LDTs as they are needed. This provides maximum flexibility for handling a dynamic programming environment. However, many operating systems use a single LDT for all tasks, allocating GDT entries in advance. An embedded system, such as a process controller, might pre-allocate a fixed number of segments and LDTs for a fixed number of application programs. This would be a simple and efficient way to structure the software environment of a real-time system.

12.8.2 Initializing Protected-Mode Exceptions and Interrupts

Software initialization code must at a minimum load a protected-mode IDT with gate descriptor for each exception vector that the processor can generate. If interrupt or trap gates are used, the gate descriptors can all point to the same code segment, which contains the necessary exception handlers. If task gates are used, one TSS and accompanying code, data, and task segments are required for each exception handler called with a task gate.

If hardware allows interrupts to be generated, gate descriptors must be provided in the IDT for one or more interrupt handlers.

Before the IDT can be used, the base address and limit for the IDT must be loaded into the IDTR register using an LIDT instruction. This operation is typically carried out immediately after switching to protected mode.

12.8.3 Initializing Paging

Paging is controlled by the PG flag in control register CR0. When this flag is clear (its state following a hardware reset), the paging mechanism is turned off; when it is set, paging is enabled. Before setting the PG flag, the following data structures and registers must be initialized:

- Software must load at least one page directory and one page table into physical memory. The page table can be eliminated if the page directory contains a directory entry pointing to itself (here, the page directory and page table reside in the same page), or if only 4-MByte pages are used.
- Control register CR3 (also called the PDBR register) is loaded with the physical base address of the page directory.
- (Optional) Software may provide one set of code and data descriptors in the GDT or in an LDT for supervisor mode and another set for user mode.

With this paging initialization complete, paging is enabled and the processor is switched to protected mode at the same time by loading control register CR0 with an image in which the PG and PE flags are set. (Paging cannot be enabled before the processor is switched to protected mode.)

12.8.4 Initializing Multitasking

If the multitasking mechanism is not going to be used and changes between privilege levels are not allowed, it is not necessary to load a TSS into memory or to initialize the task register.

If the multitasking mechanism is going to be used and/or changes between privilege levels are allowed, software initialization code must load at least one TSS and an accompanying TSS descriptor. (A TSS is required to change privilege levels because pointers to the privileged-level 0, 1, and 2 stack segments and the stack pointers for these stacks are obtained from the TSS.) TSS descriptors must not be marked as busy when they are created; they should be marked busy by the processor only as a side-effect of performing a task switch. As with descriptors for LDTs, TSS descriptors reside in the GDT.

After the processor has switched to protected mode, the LTR instruction can be used to load a segment selector for a TSS descriptor into the task register. This instruction marks the TSS descriptor as busy, but does not perform a task switch. The processor can, however, use the TSS to locate pointers to privilege-level 0, 1, and 2 stacks. The segment selector for the TSS must be loaded before software performs its first task switch in protected mode, because a task switch copies the current task state into the TSS.

After the LTR instruction has been executed, further operations on the task register are performed by task switching. As with other segments and LDTs, TSSs and TSS descriptors can be either pre-allocated or allocated as needed.

12.8.5 Initializing IA-32e Mode

On Intel 64 processors, the IA32_EFER MSR is cleared on system reset. The operating system must be in protected mode with paging enabled before attempting to initialize IA-32e mode. IA-32e mode operation also requires physical-address extensions with four or five levels of enhanced paging structures (see Section 5.5, “4-Level Paging and 5-Level Paging”).

Operating systems should follow this sequence to initialize IA-32e mode:

1. Starting from protected mode, disable paging by setting CR0.PG = 0. Use the MOV CR0 instruction to disable paging (the instruction must be located in an identity-mapped page).
2. Enable physical-address extensions (PAE) by setting CR4.PAE = 1. Failure to enable PAE will result in a #GP fault when an attempt is made to initialize IA-32e mode.
3. Load CR3 with the physical base address of the Level 4 page map table (PML4) or Level 5 page map table (PML5).
4. Enable IA-32e mode by setting IA32_EFER.LME = 1.
5. Enable paging by setting CR0.PG = 1. This causes the processor to set the IA32_EFER.LMA bit to 1. The MOV CR0 instruction that enables paging and the following instructions must be located in an identity-mapped page (until such time that a branch to non-identity mapped pages can be effected).

64-bit mode paging structures must be located in the first 4 GBytes of physical-address space prior to activating IA-32e mode. This is necessary because the MOV CR3 instruction used to initialize the page-directory base must be executed in legacy mode prior to activating IA-32e mode (setting CR0.PG = 1 to enable paging). Because MOV CR3 is executed in protected mode, only the lower 32 bits of the register are written, limiting the table location to the low 4 GBytes of memory. Software can relocate the page tables anywhere in physical memory after IA-32e mode is activated.

The processor performs 64-bit mode consistency checks whenever software attempts to modify any of the enable bits directly involved in activating IA-32e mode (IA32_EFER.LME, CR0.PG, and CR4.PAE). It will generate a general protection fault (#GP) if consistency checks fail. 64-bit mode consistency checks ensure that the processor does not enter an undefined mode or state with unpredictable behavior.

64-bit mode consistency checks fail in the following circumstances:

- An attempt is made to enable or disable IA-32e mode while paging is enabled.
- IA-32e mode is enabled and an attempt is made to enable paging prior to enabling physical-address extensions (PAE).
- IA-32e mode is active and an attempt is made to disable physical-address extensions (PAE).
- If the current CS has the L-bit set on an attempt to activate IA-32e mode.
- If the TR contains a 16-bit TSS on an attempt to activate IA-32e mode.

12.8.5.1 IA-32e Mode System Data Structures

After activating IA-32e mode, the system-descriptor-table registers (GDTR, LDTR, IDTR, TR) continue to reference legacy protected-mode descriptor tables. Tables referenced by the descriptors all reside in the lower 4 GBytes of linear-address space. After activating IA-32e mode, 64-bit operating-systems should use the LGDT, LLDT, LIDT, and LTR instructions to load the system-descriptor-table registers with references to 64-bit descriptor tables.

12.8.5.2 IA-32e Mode Interrupts and Exceptions

Software must not allow exceptions or interrupts to occur between the time IA-32e mode is activated and the update of the interrupt-descriptor-table register (IDTR) that establishes references to a 64-bit interrupt-descriptor table (IDT). This is because the IDT remains in legacy form immediately after IA-32e mode is activated.

If an interrupt or exception occurs prior to updating the IDTR, a legacy 32-bit interrupt gate will be referenced and interpreted as a 64-bit interrupt gate with unpredictable results. External interrupts can be disabled by using the CLI instruction.

Non-maskable interrupts (NMI) must be disabled using external hardware.

12.8.5.3 64-bit Mode and Compatibility Mode Operation

IA-32e mode uses two code segment-descriptor bits (CS.L and CS.D, see Figure 3-8) to control the operating modes after IA-32e mode is initialized. If CS.L = 1 and CS.D = 0, the processor is running in 64-bit mode. With this encoding, the default operand size is 32 bits and default address size is 64 bits. Using instruction prefixes, operand size can be changed to 64 bits or 16 bits; address size can be changed to 32 bits.

When IA-32e mode is active and CS.L = 0, the processor operates in compatibility mode. In this mode, CS.D controls default operand and address sizes exactly as it does in the IA-32 architecture. Setting CS.D = 1 specifies default operand and address size as 32 bits. Clearing CS.D to 0 specifies default operand and address size as 16 bits (the CS.L = 1, CS.D = 1 bit combination is reserved).

Compatibility mode execution is selected on a code-segment basis. This mode allows legacy applications to coexist with 64-bit applications running in 64-bit mode. An operating system running in IA-32e mode can execute existing 16-bit and 32-bit applications by clearing their code-segment descriptor's CS.L bit to 0.

In compatibility mode, the following system-level mechanisms continue to operate using the IA-32e-mode architectural semantics:

- Linear-to-physical address translation uses the 64-bit mode extended page-translation mechanism.
- Interrupts and exceptions are handled using the 64-bit mode mechanisms.
- System calls (calls through call gates and SYSENTER/SYSEXIT) are handled using the IA-32e mode mechanisms.

12.8.5.4 Switching Out of IA-32e Mode Operation

To return from IA-32e mode to paged-protected mode operation operating systems must use the following sequence:

1. Switch to compatibility mode.
2. Deactivate IA-32e mode by clearing CR0.PG = 0. This causes the processor to set IA32_EFER.LMA = 0. The MOV CR0 instruction used to disable paging and subsequent instructions must be located in an identity-mapped page.
3. Load CR3 with the physical base address of the legacy page-table-directory base address.
4. Disable IA-32e mode by setting IA32_EFER.LME = 0.
5. Enable legacy paged-protected mode by setting CR0.PG = 1
6. A branch instruction must follow the MOV CR0 that enables paging. Both the MOV CR0 and the branch instruction must be located in an identity-mapped page.

Registers only available in 64-bit mode (R8-R15 and XMM8-XMM15) are preserved across transitions from 64-bit mode into compatibility mode then back into 64-bit mode. However, values of R8-R15 and XMM8-XMM15 are undefined after transitions from 64-bit mode through compatibility mode to legacy or real mode and then back through compatibility mode to 64-bit mode.

12.9 MODE SWITCHING

To use the processor in protected mode after hardware or software reset, a mode switch must be performed from real-address mode. Once in protected mode, software generally does not need to return to real-address mode. To run software written to run in real-address mode (8086 mode), it is generally more convenient to run the software in virtual-8086 mode, than to switch back to real-address mode.

12.9.1 Switching to Protected Mode

Before switching to protected mode from real mode, a minimum set of system data structures and code modules must be loaded into memory, as described in Section 12.8, “Software Initialization for Protected-Mode Operation.” Once these tables are created, software initialization code can switch into protected mode.

Protected mode is entered by executing a MOV CR0 instruction that sets the PE flag in the CR0 register. (In the same instruction, the PG flag in register CR0 can be set to enable paging.) Execution in protected mode begins with a CPL of 0.

Intel 64 and IA-32 processors have slightly different requirements for switching to protected mode. To ensure upwards and downwards code compatibility with Intel 64 and IA-32 processors, we recommend that you follow these steps:

1. Disable interrupts. A CLI instruction disables maskable hardware interrupts. NMI interrupts can be disabled with external circuitry. (Software must guarantee that no exceptions or interrupts are generated during the mode switching operation.)
2. Execute the LGDT instruction to load the GDTR register with the base address of the GDT.
3. Execute a MOV CR0 instruction that sets the PE flag (and optionally the PG flag) in control register CR0.
4. Immediately following the MOV CR0 instruction, execute a far JMP or far CALL instruction. (This operation is typically a far jump or call to the next instruction in the instruction stream.)
5. The JMP or CALL instruction immediately after the MOV CR0 instruction changes the flow of execution and serializes the processor.
6. If paging is enabled, the code for the MOV CR0 instruction and the JMP or CALL instruction must come from a page that is identity mapped (that is, the linear address before the jump is the same as the physical address after paging and protected mode is enabled). The target instruction for the JMP or CALL instruction does not need to be identity mapped.
7. If a local descriptor table is going to be used, execute the LLDT instruction to load the segment selector for the LDT in the LDTR register.
8. Execute the LTR instruction to load the task register with a segment selector to the initial protected-mode task or to a writable area of memory that can be used to store TSS information on a task switch.
9. After entering protected mode, the segment registers continue to hold the contents they had in real-address mode. The JMP or CALL instruction in step 4 resets the CS register. Perform one of the following operations to update the contents of the remaining segment registers.
 - Reload segment registers DS, SS, ES, FS, and GS. If the ES, FS, and/or GS registers are not going to be used, load them with a null selector.
 - Perform a JMP or CALL instruction to a new task, which automatically resets the values of the segment registers and branches to a new code segment.
10. Execute the LIDT instruction to load the IDTR register with the address and limit of the protected-mode IDT.
11. Execute the STI instruction to enable maskable hardware interrupts and perform the necessary hardware operation to enable NMI interrupts.

Random failures can occur if other instructions exist between steps 3 and 4 above. Failures will be readily seen in some situations, such as when instructions that reference memory are inserted between steps 3 and 4 while in system management mode.

12.9.2 Switching Back to Real-Address Mode

The processor switches from protected mode back to real-address mode if software clears the PE bit in the CR0 register with a MOV CR0 instruction. A procedure that re-enters real-address mode should perform the following steps:

1. Disable interrupts. A CLI instruction disables maskable hardware interrupts. NMI interrupts can be disabled with external circuitry.
2. If paging is enabled, perform the following operations:
 - Transfer program control to linear addresses that are identity mapped to physical addresses (that is, linear addresses equal physical addresses).
 - Ensure that the GDT and IDT are in identity mapped pages.
 - Clear the PG bit in the CR0 register.
 - Move 0H into the CR3 register to flush the TLB.
3. Transfer program control to a readable segment that has a limit of 64 KBytes (FFFFH). This operation loads the CS register with the segment limit required in real-address mode.
4. Load segment registers SS, DS, ES, FS, and GS with a selector for a descriptor containing the following values, which are appropriate for real-address mode:
 - Limit = 64 KBytes (0FFFFH)
 - Byte granular (G = 0)
 - Expand up (E = 0)
 - Writable (W = 1)
 - Present (P = 1)
 - Base = any value

The segment registers must be loaded with non-null segment selectors or the segment registers will be unusable in real-address mode. Note that if the segment registers are not reloaded, execution continues using the descriptor attributes loaded during protected mode.

5. Execute an LIDT instruction to point to a real-address mode interrupt table that is within the 1-MByte real-address mode address range.
6. Clear the PE flag in the CR0 register to switch to real-address mode.
7. Execute a far JMP instruction to jump to a real-address mode program. This operation flushes the instruction queue and loads the appropriate base-address value in the CS register.
8. Load the SS, DS, ES, FS, and GS registers as needed by the real-address mode code. If any of the registers are not going to be used in real-address mode, write 0s to them.
9. Execute the STI instruction to enable maskable hardware interrupts and perform the necessary hardware operation to enable NMI interrupts.

NOTE

All the code that is executed in steps 1 through 9 must be in a single page and the linear addresses in that page must be identity mapped to physical addresses.

12.10 INITIALIZATION AND MODE SWITCHING EXAMPLE

This section provides an initialization and mode switching example that can be incorporated into an application. This code was originally written to initialize the Intel386 processor, but it will execute successfully on the Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors. The code in this example is intended to reside in EPROM and to run following a hardware reset of the processor. The function of the code is to do the following:

- Establish a basic real-address mode operating environment.

- Load the necessary protected-mode system data structures into RAM.
- Load the system registers with the necessary pointers to the data structures and the appropriate flag settings for protected-mode operation.
- Switch the processor to protected mode.

Figure 12-3 shows the physical memory layout for the processor following a hardware reset and the starting point of this example. The EPROM that contains the initialization code resides at the upper end of the processor's physical memory address range, starting at address FFFFFFFFH and going down from there. The address of the first instruction to be executed is at FFFFFFF0H, the default starting address for the processor following a hardware reset.

The main steps carried out in this example are summarized in Table 12-5. The source listing for the example (with the filename STARTUP.ASM) is given in Example 12-1. The line numbers given in Table 12-5 refer to the source listing.

The following are some additional notes concerning this example:

- When the processor is switched into protected mode, the original code segment base-address value of FFFF0000H (located in the hidden part of the CS register) is retained and execution continues from the current offset in the EIP register. The processor will thus continue to execute code in the EPROM until a far jump or call is made to a new code segment, at which time, the base address in the CS register will be changed.
- Maskable hardware interrupts are disabled after a hardware reset and should remain disabled until the necessary interrupt handlers have been installed. The NMI interrupt is not disabled following a reset. The NMI# pin must thus be inhibited from being asserted until an NMI handler has been loaded and made available to the processor.
- The use of a temporary GDT allows simple transfer of tables from the EPROM to anywhere in the RAM area. A GDT entry is constructed with its base pointing to address 0 and a limit of 4 GBytes. When the DS and ES registers are loaded with this descriptor, the temporary GDT is no longer needed and can be replaced by the application GDT.
- This code loads one TSS and no LDTs. If more TSSs exist in the application, they must be loaded into RAM. If there are LDTs they may be loaded as well.

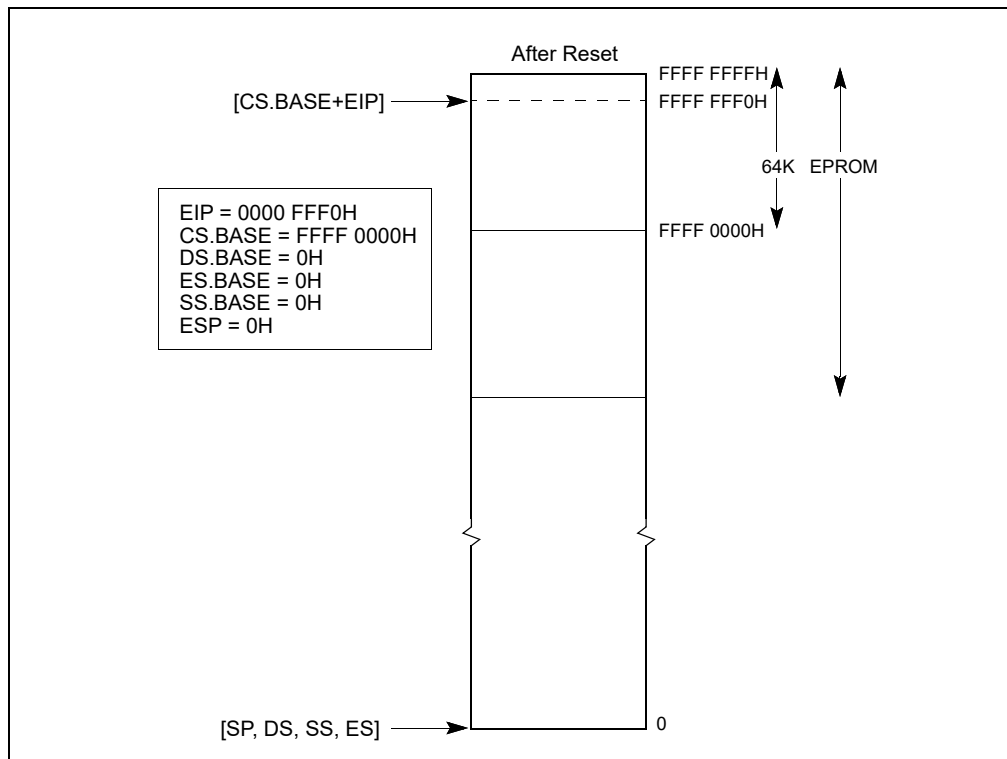


Figure 12-3. Processor State After Reset

Table 12-5. Main Initialization Steps in STARTUP.ASM Source Listing

STARTUP.ASM Line Numbers		Description
From	To	
157	157	Jump (short) to the entry code in the EPROM
162	169	Construct a temporary GDT in RAM with one entry: 0 - null 1 - R/W data segment, base = 0, limit = 4 GBytes
171	172	Load the GDTR to point to the temporary GDT
174	177	Load CR0 with PE flag set to switch to protected mode
179	181	Jump near to clear real mode instruction queue
184	186	Load DS, ES registers with GDT[1] descriptor, so both point to the entire physical memory space
188	195	Perform specific board initialization that is imposed by the new protected mode
196	218	Copy the application's GDT from ROM into RAM
220	238	Copy the application's IDT from ROM into RAM
241	243	Load application's GDTR
244	245	Load application's IDTR
247	261	Copy the application's TSS from ROM into RAM
263	267	Update TSS descriptor and other aliases in GDT (GDT alias or IDT alias)
277	277	Load the task register (without task switch) using LTR instruction
282	286	Load SS, ESP with the value found in the application's TSS
287	287	Push EFLAGS value found in the application's TSS
288	288	Push CS value found in the application's TSS
289	289	Push EIP value found in the application's TSS
290	293	Load DS, ES with the value found in the application's TSS
296	296	Perform IRET; pop the above values and enter the application code

12.10.1 Assembler Usage

In this example, the Intel assembler ASM386 and build tools BLD386 are used to assemble and build the initialization code module. The following assumptions are used when using the Intel ASM386 and BLD386 tools.

- The ASM386 will generate the right operand size opcodes according to the code-segment attribute. The attribute is assigned either by the ASM386 invocation controls or in the code-segment definition.
- If a code segment that is going to run in real-address mode is defined, it must be set to a USE 16 attribute. If a 32-bit operand is used in an instruction in this code segment (for example, MOV EAX, EBX), the assembler automatically generates an operand prefix for the instruction that forces the processor to execute a 32-bit operation, even though its default code-segment attribute is 16-bit.
- Intel's ASM386 assembler allows specific use of the 16- or 32-bit instructions, for example, LGDTW, LGDTD, IRETD. If the generic instruction LGDT is used, the default- segment attribute will be used to generate the right opcode.

12.10.2 STARTUP.ASM Listing

Example 12-1 provides high-level sample code designed to move the processor into protected mode. This listing does not include any opcode and offset information.

Example 12-1. STARTUP.ASM

MS-DOS* 5.0(045-N) 386(TM) MACRO ASSEMBLER STARTUP 09:44:51 08/19/92 PAGE 1

MS-DOS 5.0(045-N) 386(TM) MACRO ASSEMBLER V4.0, ASSEMBLY OF MODULE STARTUP
 OBJECT MODULE PLACED IN startup.obj
 ASSEMBLER INVOKED BY: f:\386tools\ASM386.EXE startup.a58 pw (132)

```

LINE      SOURCE

1          NAME      STARTUP
2
3          ;;;;;;;;;;;;;;
4          ;
5          ;  ASSUMPTIONS:
6          ;
7          ;    1.  The bottom 64K of memory is ram, and can be used for
8          ;        scratch space by this module.
9          ;
10         ;    2.  The system has sufficient free usable ram to copy the
11         ;        initial GDT, IDT, and TSS
12         ;
13         ;;;;;;;;;;;;;;
14
15         ; configuration data - must match with build definition
16
17         CS_BASE      EQU      0FFFF0000H
18
19         ; CS_BASE is the linear address of the segment STARTUP_CODE
20         ; - this is specified in the build language file
21
22         RAM_START     EQU      400H
23
24         ; RAM_START is the start of free, usable ram in the linear
25         ; memory space.  The GDT, IDT, and initial TSS will be
26         ; copied above this space, and a small data segment will be
27         ; discarded at this linear address.  The 32-bit word at
28         ; RAM_START will contain the linear address of the first
29         ; free byte above the copied tables - this may be useful if
30         ; a memory manager is used.
31
32         TSS_INDEX     EQU      10
33
34         ; TSS_INDEX is the index of the TSS of the first task to
35         ; run after startup
36
37
38         ;;;;;;;;;;;;;;
39
40         ; ----- STRUCTURES and EQU -----
41         ; structures for system data
42
43         ; TSS structure
44         TASK_STATE     STRUC
45         link           DW ?

```

PROCESSOR MANAGEMENT AND INITIALIZATION

```
46     link_h    DW ?
47     ESP0      DD ?
48     SS0       DW ?
49     SS0_h     DW ?
50     ESP1      DD ?
51     SS1       DW ?
52     SS1_h     DW ?
53     ESP2      DD ?
54     SS2       DW ?
55     SS2_h     DW ?
56     CR3_reg   DD ?
57     EIP_reg   DD ?
58     EFLAGS_reg DD ?
59     EAX_reg   DD ?
60     ECX_reg   DD ?
61     EDX_reg   DD ?
62     EBX_reg   DD ?
63     ESP_reg   DD ?
64     EBP_reg   DD ?
65     ESI_reg   DD ?
66     EDI_reg   DD ?
67     ES_reg    DW ?
68     ES_h      DW ?
69     CS_reg    DW ?
70     CS_h      DW ?
71     SS_reg    DW ?
72     SS_h      DW ?
73     DS_reg    DW ?
74     DS_h      DW ?
75     FS_reg    DW ?
76     FS_h      DW ?
77     GS_reg    DW ?
78     GS_h      DW ?
79     LDT_reg   DW ?
80     LDT_h     DW ?
81     TRAP_reg  DW ?
82     IO_map_base DW ?
83 TASK_STATE  ENDS
84
85 ; basic structure of a descriptor
86 DESC        STRUC
87     lim_0_15 DW ?
88     bas_0_15 DW ?
89     bas_16_23 DB ?
90     access   DB ?
91     gran     DB ?
92     bas_24_31 DB ?
93 DESC        ENDS
94
95 ; structure for use with LGDT and LIDT instructions
96 TABLE_REG   STRUC
97     table_lim DW ?
98     table_linear DD ?
99 TABLE_REG   ENDS
```

```

100
101 ; offset of GDT and IDT descriptors in builder generated GDT
102 GDT_DESC_OFF EQU 1*SIZE(DESC)
103 IDT_DESC_OFF EQU 2*SIZE(DESC)
104
105 ; equates for building temporary GDT in RAM
106 LINEAR_SEL EQU 1*SIZE(DESC)
107 LINEAR_PROTO_LO EQU 00000FFFFH ; LINEAR_ALIAS
108 LINEAR_PROTO_HI EQU 000CF9200H
109
110 ; Protection Enable Bit in CR0
111 PE_BIT EQU 1B
112
113 ; -----
114
115 ; ----- DATA SEGMENT-----
116
117 ; Initially, this data segment starts at linear 0, according
118 ; to the processor's power-up state.
119
120 STARTUP_DATA SEGMENT RW
121
122 free_mem_linear_base LABEL DWORD
123 TEMP_GDT LABEL BYTE ; must be first in segment
124 TEMP_GDT_NULL_DESC DESC <>
125 TEMP_GDT_LINEAR_DESC DESC <>
126
127 ; scratch areas for LGDT and LIDT instructions
128 TEMP_GDT_SCRATCH TABLE_REG <>
129 APP_GDT_RAM TABLE_REG <>
130 APP_IDT_RAM TABLE_REG <>
131 ; align end_data
132 fill DW ?
133
134 ; last thing in this segment - should be on a dword boundary
135 end_data LABEL BYTE
136
137 STARTUP_DATA ENDS
138 ; -----
139
140
141 ; ----- CODE SEGMENT-----
142 STARTUP_CODE SEGMENT ER PUBLIC USE16
143
144 ; filled in by builder
145 PUBLIC GDT_EPROM
146 GDT_EPROM TABLE_REG <>
147
148 ; filled in by builder
149 PUBLIC IDT_EPROM
150 IDT_EPROM TABLE_REG <>
151
152 ; entry point into startup code - the bootstrap will vector
153 ; here with a near JMP generated by the builder. This

```

```

154 ; label must be in the top 64K of linear memory.
155
156     PUBLIC  STARTUP
157 STARTUP:
158
159 ; DS,ES address the bottom 64K of flat linear memory
160     ASSUME  DS:STARTUP_DATA, ES:STARTUP_DATA
161 ; See Figure 12-4
162 ; load GDTR with temporary GDT
163     LEA     EBX,TEMP_GDT ; build the TEMP_GDT in low ram,
164     MOV     DWORD PTR [EBX],0 ; where we can address
165     MOV     DWORD PTR [EBX]+4,0
166     MOV     DWORD PTR [EBX]+8, LINEAR_PROTO_LO
167     MOV     DWORD PTR [EBX]+12, LINEAR_PROTO_HI
168     MOV     TEMP_GDT_scratch.table_linear,EBX
169     MOV     TEMP_GDT_scratch.table_lim,15
170
171     DB 66H; execute a 32 bit LGDT
172     LGDT    TEMP_GDT_scratch
173
174 ; enter protected mode
175     MOV     EBX,CR0
176     OR      EBX,PE_BIT
177     MOV     CR0,EBX
178
179 ; clear prefetch queue
180     JMP     CLEAR_LABEL
181 CLEAR_LABEL:
182
183 ; make DS and ES address 4G of linear memory
184     MOV     CX,LINEAR_SEL
185     MOV     DS,CX
186     MOV     ES,CX
187
188 ; do board specific initialization
189 ;
190 ;
191 ; .....
192 ;
193
194
195 ; See Figure 12-5
196 ; copy EPROM GDT to ram at:
197 ;             RAM_START + size (STARTUP_DATA)
198     MOV     EAX,RAM_START
199     ADD     EAX,OFFSET (end_data)
200     MOV     EBX,RAM_START
201     MOV     ECX, CS_BASE
202     ADD     ECX, OFFSET (GDT_EPROM)
203     MOV     ESI, [ECX].table_linear
204     MOV     EDI,EAX
205     MOVZX   ECX, [ECX].table_lim
206     MOV     APP_GDT_ram[EBX].table_lim,CX

```

```

207     INC     ECX
208     MOV     EDX,EAX
209     MOV     APP_GDT_ram[EBX].table_linear,EAX
210     ADD     EAX,ECX
211     REP MOVSB     BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
212
213     ; fixup GDT base in descriptor
214     MOV     ECX,EDX
215     MOV     [EDX].bas_0_15+GDT_DESC_OFF,CX
216     ROR     ECX,16
217     MOV     [EDX].bas_16_23+GDT_DESC_OFF,CL
218     MOV     [EDX].bas_24_31+GDT_DESC_OFF,CH
219
220     ; copy EPROM IDT to ram at:
221     ; RAM_START+size(STARTUP_DATA)+SIZE (EPROM GDT)
222     MOV     ECX, CS_BASE
223     ADD     ECX, OFFSET (IDT_EPROM)
224     MOV     ESI, [ECX].table_linear
225     MOV     EDI,EAX
226     MOVZX   ECX, [ECX].table_lim
227     MOV     APP_IDT_ram[EBX].table_lim,CX
228     INC     ECX
229     MOV     APP_IDT_ram[EBX].table_linear,EAX
230     MOV     EBX,EAX
231     ADD     EAX,ECX
232     REP MOVSB     BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
233
234     ; fixup IDT pointer in GDT
235     MOV     [EDX].bas_0_15+IDT_DESC_OFF,BX
236     ROR     EBX,16
237     MOV     [EDX].bas_16_23+IDT_DESC_OFF,BL
238     MOV     [EDX].bas_24_31+IDT_DESC_OFF,BH
239
240     ; load GDTR and IDTR
241     MOV     EBX, RAM_START
242     DB      66H                ; execute a 32 bit LGDT
243     LGDT    APP_GDT_ram[EBX]
244     DB      66H                ; execute a 32 bit LIDT
245     LIDT    APP_IDT_ram[EBX]
246
247     ; move the TSS
248     MOV     EDI,EAX
249     MOV     EBX,TSS_INDEX*SIZE(DESC)
250     MOV     ECX,GDT_DESC_OFF ;build linear address for TSS
251     MOV     GS,CX
252     MOV     DH,GS:[EBX].bas_24_31
253     MOV     DL,GS:[EBX].bas_16_23
254     ROL     EDX,16
255     MOV     DX,GS:[EBX].bas_0_15
256     MOV     ESI,EDX
257     LSL     ECX,EBX
258     INC     ECX
259     MOV     EDX,EAX
260     ADD     EAX,ECX

```


PROCESSOR MANAGEMENT AND INITIALIZATION

```
261      REP MOVSB    BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
262
263          ; fixup TSS pointer
264      MOV     GS:[EBX].bas_0_15,DX
265      ROL     EDX,16
266      MOV     GS:[EBX].bas_24_31,DH
267      MOV     GS:[EBX].bas_16_23,DL
268      ROL     EDX,16
269      ;save start of free ram at linear location RAMSTART
270      MOV     free_mem_linear_base+RAM_START,EAX
271
272      ;assume no LDT used in the initial task - if necessary,
273      ;code to move the LDT could be added, and should resemble
274      ;that used to move the TSS
275
276      ; load task register
277      LTR     BX    ; No task switch, only descriptor loading
278      ; See Figure 12-6
279      ; load minimal set of registers necessary to simulate task
280      ; switch
281
282
283      MOV     AX,[EDX].SS_reg    ; start loading registers
284      MOV     EDI,[EDX].ESP_reg
285      MOV     SS,AX
286      MOV     ESP,EDI           ; stack now valid
287      PUSH    DWORD PTR [EDX].EFLAGS_reg
288      PUSH    DWORD PTR [EDX].CS_reg
289      PUSH    DWORD PTR [EDX].EIP_reg
290      MOV     AX,[EDX].DS_reg
291      MOV     BX,[EDX].ES_reg
292      MOV     DS,AX             ; DS and ES no longer linear memory
293      MOV     ES,BX
294
295      ; simulate far jump to initial task
296      IRETD
297
298  STARTUP_CODE  ENDS
*** WARNING #377 IN 298, (PASS 2) SEGMENT CONTAINS PRIVILEGED INSTRUCTION(S)
299
300  END STARTUP, DS:STARTUP_DATA, SS:STARTUP_DATA
301
302
```

ASSEMBLY COMPLETE, 1 WARNING, NO ERRORS.

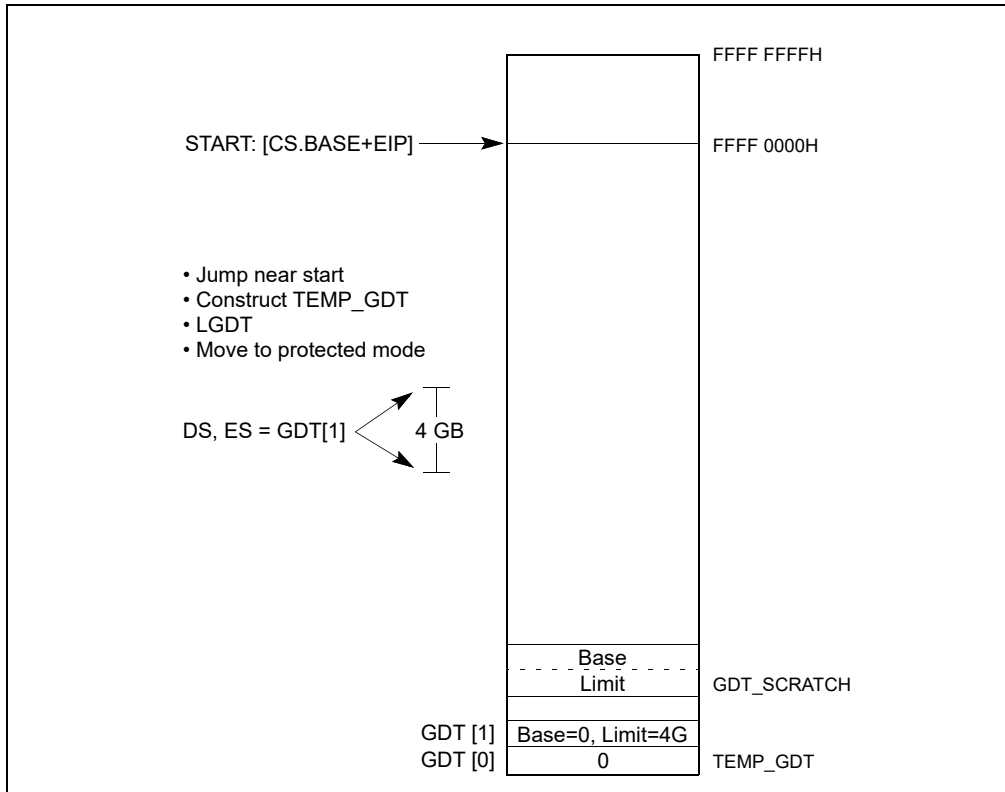


Figure 12-4. Constructing Temporary GDT and Switching to Protected Mode (Lines 162-172 of List File)

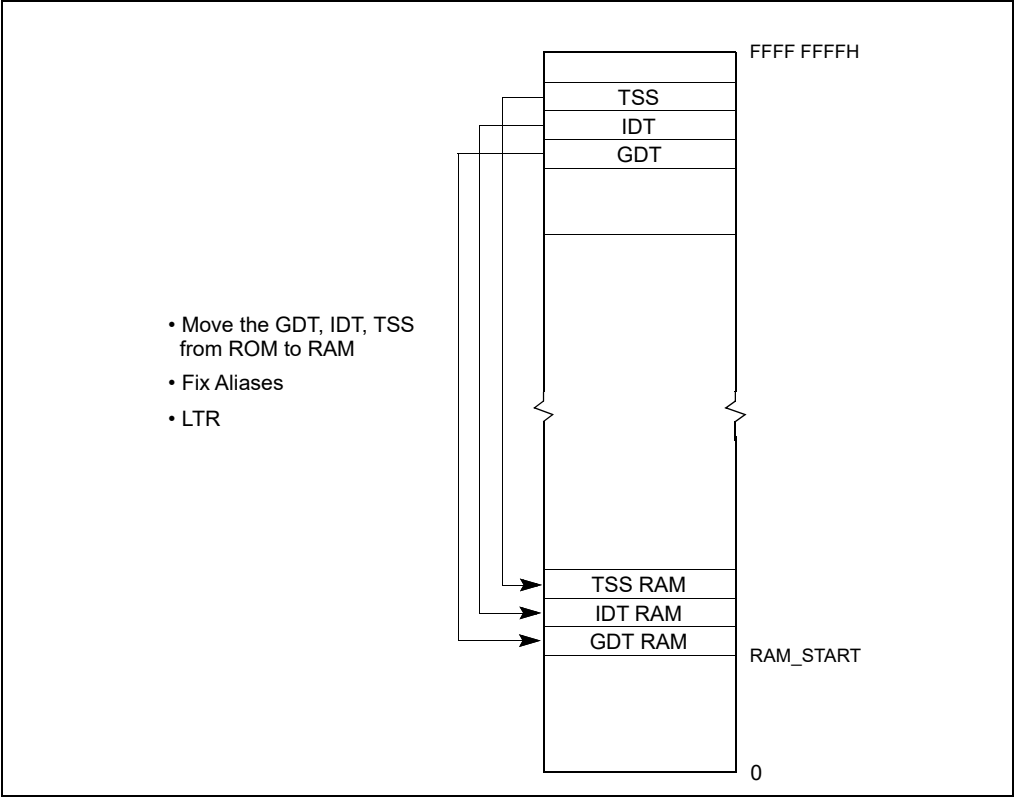


Figure 12-5. Moving the GDT, IDT, and TSS from ROM to RAM (Lines 196-261 of List File)

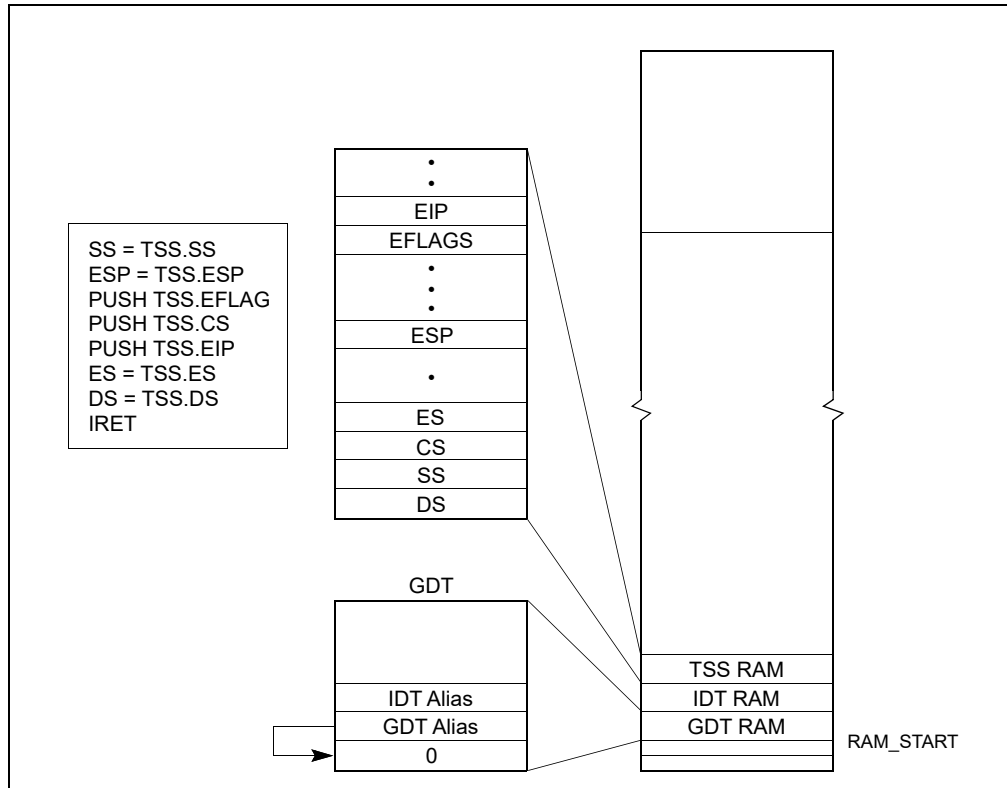


Figure 12-6. Task Switching (Lines 282-296 of List File)

12.10.3 MAIN.ASM Source Code

The file MAIN.ASM shown in Example 12-2 defines the data and stack segments for this application and can be substituted with the main module task written in a high-level language that is invoked by the IRET instruction executed by STARTUP.ASM.

Example 12-2. MAIN.ASM

```
NAME    main_module
data    SEGMENT RW
        dw 1000 dup(?)
DATA    ENDS
stack   stackseg 800
CODE SEGMENT ER use32 PUBLIC
main_start:
        nop
        nop
        nop
CODE    ENDS
END main_start, ds:data, ss:stack
```

12.10.4 Supporting Files

The batch file shown in Example 12-3 can be used to assemble the source code files STARTUP.ASM and MAIN.ASM and build the final application.

Example 12-3. Batch File to Assemble and Build the Application

```

ASM386 STARTUP.ASM
ASM386 MAIN.ASM
BLD386 STARTUP.OBJ, MAIN.OBJ buildfile(EPROM.BLD) bootstrap(STARTUP) Bootload

```

BLD386 performs several operations in this example:
 It allocates physical memory location to segments and tables.
 It generates tables using the build file and the input files.
 It links object files and resolves references.
 It generates a boot-loadable file to be programmed into the EPROM.

Example 12-4 shows the build file used as an input to BLD386 to perform the above functions.

Example 12-4. Build File

```

INIT_BLD_EXAMPLE;

SEGMENT
    *SEGMENTS(DPL = 0)
    ,   startup.startup_code(BASE = 0FFFF0000H)
    ;

TASK
    BOOT_TASK(OBJECT = startup, INITIAL,DPL = 0,
              NOT INTENABLED)
    ,   PROTECTED_MODE_TASK(OBJECT = main_module,DPL = 0,
              NOT INTENABLED)
    ;

TABLE
    GDT (
        LOCATION = GDT_EPROM
        ,   ENTRY = (
            10:   PROTECTED_MODE_TASK
            ,   startup.startup_code
            ,   startup.startup_data
            ,   main_module.data
            ,   main_module.code
            ,   main_module.stack
            )
        ),

    IDT (
        LOCATION = IDT_EPROM
        );

MEMORY
    (
        RESERVE = (0..3FFFH
                    -- Area for the GDT, IDT, TSS copied from ROM
            ,   60000H..0FFFFFFFHH)
        ,   RANGE = (ROM_AREA = ROM (0FFFF0000H..0FFFFFFFHH)
                    -- Eprom size 64K
            ,   RANGE = (RAM_AREA = RAM (4000H..05FFFFH))
        )
    )

```

);

END

Table 12-6 shows the relationship of each build item with an ASM source file.

Table 12-6. Relationship Between BLD Item and ASM Source File

Item	ASM386 and Startup.A58	BLD386 Controls and BLD file	Effect
Bootstrap	public startup startup:	bootstrap start(startup)	Near jump at 0FFFFFF0H to start.
GDT location	public GDT_EEPROM GDT_EEPROM TABLE_REG <>	TABLE GDT(location = GDT_EEPROM)	The location of the GDT will be programmed into the GDT_EEPROM location.
IDT location	public IDT_EEPROM IDT_EEPROM TABLE_REG <>	TABLE IDT(location = IDT_EEPROM)	The location of the IDT will be programmed into the IDT_EEPROM location.
RAM start	RAM_START equ 400H	memory (reserve = (0..3FFFH))	RAM_START is used as the ram destination for moving the tables. It must be excluded from the application's segment area.
Location of the application TSS in the GDT	TSS_INDEX EQU 10	TABLE GDT(ENTRY = (10: PROTECTED_MODE_TASK))	Put the descriptor of the application TSS in GDT entry 10.
EPROM size and location	size and location of the initialization code	SEGMENT startup.code (base = 0FFFF0000H) ...memory (RANGE(ROM_AREA = ROM(x..y))	Initialization code size must be less than 64K and resides at upper most 64K of the 4-GByte memory space.

12.11 MICROCODE UPDATE FACILITIES

The P6 family and later processors have the capability to correct errata by loading an Intel-supplied data block into the processor. The data block is called a microcode update. This section describes the mechanisms the BIOS needs to provide in order to use this feature during system initialization. It also describes a specification that permits the incorporation of future updates into a system BIOS.

Intel considers the release of a microcode update for a silicon revision to be the equivalent of a processor stepping and completes a full-stepping level validation for releases of microcode updates.

A microcode update is used to correct errata in the processor. The BIOS, which has an update loader, is responsible for loading the update on processors during system initialization (Figure 12-7). There are two steps to this process: the first is to incorporate the necessary update data blocks into the BIOS; the second is to load update data blocks into the processor.

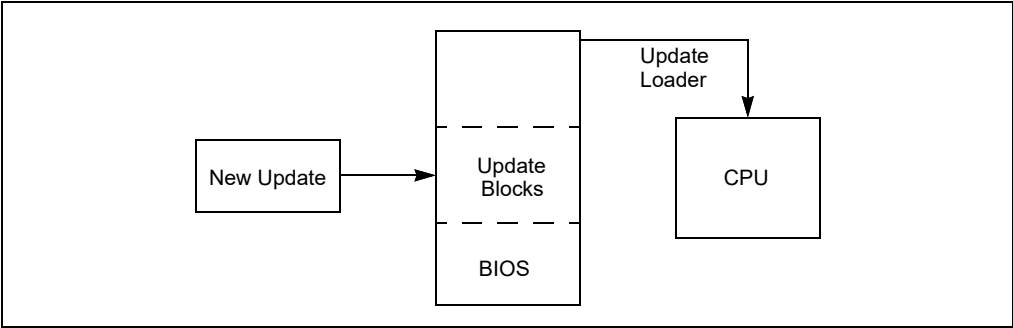


Figure 12-7. Applying Microcode Updates

12.11.1 Microcode Update

A microcode update consists of an Intel-supplied binary that contains a descriptive header and data. No executable code resides within the update. Each microcode update is tailored for a specific list of processor signatures. A mismatch of the processor’s signature with the signature contained in the update will result in a failure to load. A processor signature includes the extended family, extended model, type, family, model, and stepping of the processor (starting with processor family 0fH, model 03H, a given microcode update may be associated with one of multiple processor signatures; see Section 12.11.2 for details).

Microcode updates are composed of a multi-byte header, followed by encrypted data and then by an optional extended signature table. Table 12-7 provides a definition of the fields; Table 12-8 shows the format of an update.

The header is 48 bytes. The first 4 bytes of the header contain the header version. The update header and its reserved fields are interpreted by software based upon the header version. An encoding scheme guards against tampering and provides a means for determining the authenticity of any given update. For microcode updates with a data size field equal to 00000000H, the size of the microcode update is 2048 bytes. The first 48 bytes contain the microcode update header. The remaining 2000 bytes contain encrypted data.

For microcode updates with a data size not equal to 00000000H, the total size field specifies the size of the microcode update. The first 48 bytes contain the microcode update header. The second part of the microcode update is the encrypted data. The data size field of the microcode update header specifies the encrypted data size, its value must be a multiple of the size of DWORD. The total size field of the microcode update header specifies the encrypted data size plus the header size; its value must be in multiples of 1024 bytes (1 KBytes). The optional extended signature table if implemented follows the encrypted data, and its size is calculated by (Total Size – (Data Size + 48)).

NOTE

The optional extended signature table is supported starting with processor family 0FH, model 03H.

Table 12-7. Microcode Update Field Definitions

Field Name	Offset (bytes)	Length (bytes)	Description
Header Version	0	4	Version number of the update header.
Update Revision	4	4	Unique version number for the update, the basis for the update signature provided by the processor to indicate the current update functioning within the processor. Used by the BIOS to authenticate the update and verify that the processor loads successfully. The value in this field cannot be used for processor stepping identification alone. This is a signed 32-bit number.
Date	8	4	Date of the update creation in binary format: mmddyyyy (e.g., 07/18/98 is 07181998H).

Table 12-7. Microcode Update Field Definitions (Contd.)

Field Name	Offset (bytes)	Length (bytes)	Description
Processor Signature	12	4	<i>Extended family, extended model, type, family, model, and stepping</i> of processor that requires this particular update revision (e.g., 00000650H). Each microcode update is designed specifically for a given extended family, extended model, <i>type, family, model</i> , and <i>stepping</i> of the processor. Software should use the processor signature field in conjunction with the CPUID instruction to determine whether or not an update is appropriate to load on a processor. The information encoded within this field exactly corresponds to the bit representations returned by the CPUID instruction.
Checksum	16	4	Checksum of Update Data and Header. Used to verify the integrity of the update header and data. Checksum is correct when the summation of all the DWORDs (including the extended Processor Signature Table) that comprise the microcode update result in 00000000H.
Loader Revision	20	4	Version number of the loader program needed to correctly load this update. The initial version is 00000001H.
Processor Flags	24	4	Platform type information is encoded in the lower 8 bits of this 4-byte field. Each bit represents a particular platform type for a given CPUID. Software should use the processor flags field in conjunction with the platform Id bits in MSR (17H) to determine whether or not an update is appropriate to load on a processor. Multiple bits may be set representing support for multiple platform IDs.
Data Size	28	4	Specifies the size of the encrypted data in bytes, and must be a multiple of DWORDs. If this value is 00000000H, then the microcode update encrypted data is 2000 bytes (or 500 DWORDs).
Total Size	32	4	Specifies the total size of the microcode update in bytes. It is the summation of the header size, the encrypted data size and the size of the optional extended signature table. This value is always a multiple of 1024.
Reserved	36	12	Reserved fields for future expansion.
Update Data	48	Data Size or 2000	Update data.
Extended Signature Count	Data Size + 48	4	Specifies the number of extended signature structures (Processor Signature[n], processor flags[n] and checksum[n]) that exist in this microcode update.
Extended Checksum	Data Size + 52	4	Checksum of update extended processor signature table. Used to verify the integrity of the extended processor signature table. Checksum is correct when the summation of the DWORDs that comprise the extended processor signature table results in 00000000H.
Reserved	Data Size + 56	12	Reserved fields.

Table 12-7. Microcode Update Field Definitions (Contd.)

Field Name	Offset (bytes)	Length (bytes)	Description
Processor Signature[n]	Data Size + 68 + (n * 12)	4	<i>Extended family, extended model, type, family, model, and stepping</i> of processor that requires this particular update revision (e.g., 00000650H). Each microcode update is designed specifically for a given extended family, extended model, <i>type, family, model</i>, and <i>stepping</i> of the processor. Software should use the processor signature field in conjunction with the CPUID instruction to determine whether or not an update is appropriate to load on a processor. The information encoded within this field exactly corresponds to the bit representations returned by the CPUID instruction.
Processor Flags[n]	Data Size + 72 + (n * 12)	4	Platform type information is encoded in the lower 8 bits of this 4-byte field. Each bit represents a particular platform type for a given CPUID. Software should use the processor flags field in conjunction with the platform Id bits in MSR (17H) to determine whether or not an update is appropriate to load on a processor. Multiple bits may be set representing support for multiple platform IDs.
Checksum[n]	Data Size + 76 + (n * 12)	4	Used by utility software to decompose a microcode update into multiple microcode updates where each of the new updates is constructed without the optional Extended Processor Signature Table. To calculate the Checksum, substitute the Primary Processor Signature entry and the Processor Flags entry with the corresponding Extended Patch entry. Delete the Extended Processor Signature Table entries. The Checksum is correct when the summation of all DWORDs that comprise the created Extended Processor Patch results in 00000000H.

Table 12-8. Microcode Update Format

31		24		16		8		0		Bytes			
Header Version										0			
Update Revision										4			
Month: 8			Day: 8			Year: 16				8			
Processor Signature (CPUID)										12			
Res: 4		Extended Family: 8		Extended Mode: 4		Reserved: 2		Type: 2	Family: 4		Model: 4		Stepping: 4
Checksum										16			
Loader Revision										20			
Processor Flags										24			
Reserved (24 bits)						P7	P6	P5	P4		P3	P2	P1
Data Size										28			
Total Size										32			
Reserved (12 Bytes)										36			

Table 12-8. Microcode Update Format (Contd.)

31	24	16	8	0	Bytes
Update Data (Data Size bytes, or 2000 Bytes if Data Size = 00000000H)					48
Extended Signature Count 'n'					Data Size + 48
Extended Processor Signature Table Checksum					Data Size + 52
Reserved (12 Bytes)					Data Size + 56
Processor Signature[n]					Data Size + 68 + (n * 12)
Processor Flags[n]					Data Size + 72 + (n * 12)
Checksum[n]					Data Size + 76 + (n * 12)

12.11.2 Optional Extended Signature Table

The extended signature table is a structure that may be appended to the end of the encrypted data when the encrypted data only supports a single processor signature (optional case). The extended signature table will always be present when the encrypted data supports multiple processor steppings and/or models (required case).

The extended signature table consists of a 20-byte extended signature header structure, which contains the extended signature count, the extended processor signature table checksum, and 12 reserved bytes (Table 12-9). Following the extended signature header structure, the extended signature table contains 0-to-n extended processor signature structures.

Each processor signature structure consist of the processor signature, processor flags, and a checksum (Table 12-10).

The extended signature count in the extended signature header structure indicates the number of processor signature structures that exist in the extended signature table.

The extended processor signature table checksum is a checksum of all DWORDs that comprise the extended signature table. That includes the extended signature count, extended processor signature table checksum, 12 reserved bytes and the n processor signature structures. A valid extended signature table exists when the result of a DWORD checksum is 00000000H.

Table 12-9. Extended Processor Signature Table Header Structure

Extended Signature Count 'n'	Data Size + 48
Extended Processor Signature Table Checksum	Data Size + 52
Reserved (12 Bytes)	Data Size + 56

Table 12-10. Processor Signature Structure

Processor Signature[n]	Data Size + 68 + (n * 12)
Processor Flags[n]	Data Size + 72 + (n * 12)
Checksum[n]	Data Size + 76 + (n * 12)

12.11.3 Processor Identification

Each microcode update is designed to for a specific processor or set of processors. To determine the correct microcode update to load, software must ensure that one of the processor signatures embedded in the microcode update matches the 32-bit processor signature returned by the CPUID instruction when executed by the target processor with EAX = 1. Attempting to load a microcode update that does not match a processor signature embedded in the microcode update with the processor signature returned by CPUID will cause the BIOS to reject the update.

Example 12-5 shows how to check for a valid processor signature match between the processor and microcode update.

Example 12-5. Pseudo Code to Validate the Processor Signature

```
ProcessorSignature ← CPUID.01H:EAX

If (Update.HeaderVersion = 00000001h)
{
    // first check the ProcessorSignature field
    If (ProcessorSignature = Update.ProcessorSignature)
        Success

    // if extended signature is present
    Else If (Update.TotalSize > (Update.DataSize + 48))
    {
        //
        // Assume the Data Size has been used to calculate the
        // location of Update.ProcessorSignature[0].
        //

        For (N ← 0; ((N < Update.ExtendedSignatureCount) AND
            (ProcessorSignature ≠ Update.ProcessorSignature[N])); N++);

        // if the loops ended when the iteration count is
        // less than the number of processor signatures in
        // the table, we have a match
        If (N < Update.ExtendedSignatureCount)
            Success
        Else
            Fail
    }
    Else
        Fail
Else
    Fail
```

12.11.4 Platform Identification

In addition to verifying the processor signature, the intended processor platform type must be determined to properly target the microcode update. The intended processor platform type is determined by reading the IA32_PLATFORM_ID register, (MSR 17H). This 64-bit register must be read using the RDMSR instruction.

The three platform ID bits, when read as a binary coded decimal (BCD) number, indicate the bit position in the microcode update header's processor flags field associated with the installed processor. The processor flags in the 48-byte header and the processor flags field associated with the extended processor signature structures may have multiple bits set. Each set bit represents a different platform ID that the update supports.

Register Name: IA32_PLATFORM_ID
MSR Address: 017H

Access: Read Only

IA32_PLATFORM_ID is a 64-bit register accessed only when referenced as a Qword through a RDMSR instruction.

Table 12-11. Processor Flags

Bit	Descriptions
63:53	Reserved
52:50	Platform Id Bits (RO). The field gives information concerning the intended platform for the processor. See also Table 12-8.
	52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7
49:0	Reserved

To validate the platform information, software may implement an algorithm similar to the algorithms in Example 12-6.

Example 12-6. Pseudo Code Example of Processor Flags Test

```

Flag ← 1 << IA32_PLATFORM_ID[52:50]

If (Update.HeaderVersion = 00000001h)
{
    If (Update.ProcessorFlags & Flag)
    {
        Load Update
    }
    Else
    {
        //
        // Assume the Data Size has been used to calculate the
        // location of Update.ProcessorSignature[N] and a match
        // on Update.ProcessorSignature[N] has already succeeded
        //

        If (Update.ProcessorFlags[n] & Flag)
        {
            Load Update
        }
    }
}

```

12.11.5 Microcode Update Checksum

Each microcode update contains a DWORD checksum located in the update header. It is software's responsibility to ensure that a microcode update is not corrupt. To check for a corrupt microcode update, software must perform a

unsigned DWORD (32-bit) checksum of the microcode update. Even though some fields are signed, the checksum procedure treats all DWORDs as unsigned. Microcode updates with a header version equal to 00000001H must sum all DWORDs that comprise the microcode update. A valid checksum check will yield a value of 00000000H. Any other value indicates the microcode update is corrupt and should not be loaded.

The checksum algorithm shown by the pseudo code in Example 12-7 treats the microcode update as an array of unsigned DWORDs. If the data size DWORD field at byte offset 32 equals 00000000H, the size of the encrypted data is 2000 bytes, resulting in 500 DWORDs. Otherwise the microcode update size in DWORDs = (*Total Size* / 4), where the total size is a multiple of 1024 bytes (1 KBytes).

Example 12-7. Pseudo Code Example of Checksum Test

```

N ← 512

If (Update.DataSize ≠ 00000000H)
    N ← Update.TotalSize / 4

ChkSum ← 0
For (I ← 0; I < N; I++)
{
    ChkSum ← ChkSum + MicrocodeUpdate[I]
}

If (ChkSum = 00000000H)
    Success
Else
    Fail

```

12.11.6 Microcode Update Loader

This section describes an update loader used to load an update into a P6 family or later processors. It also discusses the requirements placed on the BIOS to ensure proper loading. The update loader described contains the minimal instructions needed to load an update. The specific instruction sequence that is required to load an update is dependent upon the loader revision field contained within the update header. This revision is expected to change infrequently (potentially, only when new processor models are introduced).

Example 12-8 below represents the update loader with a loader revision of 00000001H. Note that the microcode update must be aligned on a 16-byte boundary and the size of the microcode update must be 1-KByte granular.

Example 12-8. Assembly Code Example of Simple Microcode Update Loader

```

mov  ecx,79h          ; MSR to write in ECX
xor  eax,eax          ; clear EAX
xor  ebx,ebx          ; clear EBX
mov  ax,cs            ; Segment of microcode update
shl  eax,4
mov  bx,offset Update ; Offset of microcode update
add  eax,ebx          ; Linear Address of Update in EAX
add  eax,48d          ; Offset of the Update Data within the Update
xor  edx,edx          ; Zero in EDX
WRMSR                 ; microcode update trigger

```

The loader shown in Example 12-8 assumes that *update* is the address of a microcode update (header and data) embedded within the code segment of the BIOS. It also assumes that the processor is operating in real mode. The data may reside anywhere in memory, aligned on a 16-byte boundary, that is accessible by the processor within its current operating mode.

Before the BIOS executes the microcode update trigger (WRMSR) instruction, the following must be true:

- In 64-bit mode, EAX contains the lower 32-bits of the microcode update linear address. In protected mode, EAX contains the full 32-bit linear address of the microcode update.
- In 64-bit mode, EDX contains the upper 32-bits of the microcode update linear address. In protected mode, EDX equals zero.
- ECX contains 79H (address of IA32_BIOS_UPDT_TRIG).

Other requirements are:

- The addresses for the microcode update data must be in canonical form.
- If paging is enabled, the microcode update data must map that data as present.
- The microcode update data must start at a 16-byte aligned linear address.

12.11.6.1 Hard Resets in Update Loading

The effects of a loaded update are cleared from the processor upon a hard reset. Therefore, each time a hard reset is asserted during the BIOS POST, the update must be reloaded on all processors that observed the reset. The effects of a loaded update are, however, maintained across a processor INIT. There are no side effects caused by loading an update into a processor multiple times.

12.11.6.2 Update in a Multiprocessor System

A multiprocessor (MP) system requires loading each processor with update data appropriate for its CUID and platform ID bits. The BIOS is responsible for ensuring that this requirement is met and that the loader is located in a module executed by all processors in the system. If a system design permits multiple steppings of Pentium 4, Intel Xeon, and P6 family processors to exist concurrently; then the BIOS must verify individual processors against the update header information to ensure appropriate loading. Given these considerations, it is most practical to load the update during MP initialization.

12.11.6.3 Update in a System Supporting Intel Hyper-Threading Technology

Intel Hyper-Threading Technology has implications on the loading of the microcode update. The update must be loaded for each core in a physical processor. Thus, for a processor supporting Intel Hyper-Threading Technology, only one logical processor per core is required to load the microcode update. Each individual logical processor can independently load the update. However, MP initialization must provide some mechanism (e.g., a software semaphore) to force serialization of microcode update loads and to prevent simultaneous load attempts to the same core.

12.11.6.4 Update in a System Supporting Dual-Core Technology

Dual-core technology has implications on the loading of the microcode update. The microcode update facility is not shared between processor cores in the same physical package. The update must be loaded for each core in a physical processor.

If processor core supports Intel Hyper-Threading Technology, the guideline described in Section 12.11.6.3 also applies.

12.11.6.5 Update Loader Enhancements

The update loader presented in Section 12.11.6, "Microcode Update Loader," is a minimal implementation that can be enhanced to provide additional functionality. Potential enhancements are described below:

- BIOS can incorporate multiple updates to support multiple steppings of the Pentium 4, Intel Xeon, and P6 family processors. This feature provides for operating in a mixed stepping environment on an MP system and enables a user to upgrade to a later version of the processor. In this case, modify the loader to check the CUID and platform ID bits of the processor that it is running on against the available headers before loading a particular update. The number of updates is only limited by available BIOS space.
- A loader can load the update and test the processor to determine if the update was loaded correctly. See Section 12.11.7, "Update Signature and Verification."

- A loader can verify the integrity of the update data by performing a checksum on the double words of the update summing to zero. See Section 12.11.5, “Microcode Update Checksum.”
- A loader can provide power-on messages indicating successful loading of an update.

12.11.7 Update Signature and Verification

The P6 family and later processors provide capabilities to verify the authenticity of a particular update and to identify the current update revision. This section describes the model-specific extensions of processors that support this feature. The update verification method below assumes that the BIOS will only verify an update that is more recent than the revision currently loaded in the processor.

CPUID returns a value in a model specific register in addition to its usual register return values. The semantics of CPUID cause it to deposit an update ID value in the 64-bit model-specific register at address 08BH (IA32_BIOS_SIGN_ID). If no update is present in the processor, the value in the MSR remains unmodified. The BIOS must pre-load a zero into the MSR before executing CPUID. If a read of the MSR at 8BH still returns zero after executing CPUID, this indicates that no update is present.

The update ID value returned in the EDX register after RDMSR executes indicates the revision of the update loaded in the processor. This value, in combination with the CPUID value returned in the EAX register, uniquely identifies a particular update. The signature ID can be directly compared with the update revision field in a microcode update header for verification of a correct load. No consecutive updates released for a given stepping of a processor may share the same signature. The processor signature returned by CPUID differentiates updates for different step-pings.

12.11.7.1 Determining the Signature

An update that is successfully loaded into the processor provides a signature that matches the update revision of the currently functioning revision. This signature is available any time after the actual update has been loaded. Requesting the signature does not have a negative impact upon a loaded update.

The procedure for determining this signature shown in Example 12-9.

Example 12-9. Assembly Code to Retrieve the Update Revision

```
MOV    ECX, 08BH           ;IA32_BIOS_SIGN_ID
XOR     EAX, EAX           ;clear EAX
XOR     EDX, EDX           ;clear EDX
WRMSR                      ;Load 0 to MSR at 8BH
MOV     EAX, 1
cpuid
MOV     ECX, 08BH           ;IA32_BIOS_SIGN_ID
rdmsr                      ;Read Model Specific Register
```

If there is an update active in the processor, its revision is returned in the EDX register after the RDMSR instruction executes.

IA32_BIOS_SIGN_ID	Microcode Update Signature Register
MSR Address:	08BH Accessed as a Qword
Default Value:	XXXX XXXX XXXX XXXXh
Access:	Read/Write

The IA32_BIOS_SIGN_ID register is used to report the microcode update signature when CPUID executes. The signature is returned in the upper DWORD (Table 12-12).

Table 12-12. Microcode Update Signature

Bit	Description
63:32	Microcode update signature. This field contains the signature of the currently loaded microcode update when read following the execution of the CPUID.(EAX=01H). It is required that this register field be pre-loaded with zero prior to executing the CPUID, function 1. If the field remains equal to zero, then there is no microcode update loaded. Another non-zero value will be the signature.
31:0	Reserved.

12.11.7.2 Authenticating the Update

An update may be authenticated by the BIOS using the signature primitive, described above, and the algorithm in Example 12-10.

Example 12-10. Pseudo Code to Authenticate the Update

```

Z ← Obtain Update Revision from the Update Header to be authenticated;
X ← Obtain Current Update Signature from MSR 8BH;

If (Z > X)
{
    Load Update that is to be authenticated;
    Y ← Obtain New Signature from MSR 8BH;

    If (Z = Y)
        Success
    Else
        Fail
}
Else
    Fail

```

Example 12-10 requires that the BIOS only authenticate updates that contain a numerically larger revision than the currently loaded revision, where Current Signature (X) < New Update Revision (Z). A processor with no loaded update is considered to have a revision equal to zero.

This authentication procedure relies upon the decoding provided by the processor to verify an update from a potentially hostile source. As an example, this mechanism in conjunction with other safeguards provides security for dynamically incorporating field updates into the BIOS.

12.11.8 Optional Processor Microcode Update Specifications

This section describes an interface that an OEM-BIOS may provide to its client system software to manage processor microcode updates. System software may choose to build its own facility to manage microcode updates (e.g., similar to the facility described in [Section 12.11.6](#)) or rely on a facility provided by the BIOS to perform microcode updates.

For this update facility, regardless of the actual implementation, the BIOS is responsible for ensuring the validity and the integrity of the information provided by the caller, including authenticating the proposed update before incorporating it into non-volatile storage.

Sections 12.11.8.1-12.11.8.9 describes a [legacy](#) extension (Function 0D042H) to the real mode INT 15H service. INT 15H 0D042H function is one of several alternatives that a BIOS may choose to implement microcode update facility and offer to its client application (e.g., an OS).

Alternatively, BIOS can choose a [more modern microcode update facility](#) based on platform-specific capabilities, including the Capsule Update mechanism from the UEFI specification (www.uefi.org). In this discussion, the application is referred to as the calling program or caller.

The real mode INT15 call specification described here is an Intel extension to an OEM BIOS. This extension allows an application to read and modify the contents of the microcode update data in NVRAM. The update loader, which is part of the system BIOS, cannot be updated by the interface. All of the functions defined in the specification must be implemented for a system to be considered compliant with the specification. The INT15 functions are accessible only from real mode.

12.11.8.1 Responsibilities of the BIOS

If a BIOS passes the presence test (INT 15H, AX = 0D042H, BL = 0H), it must implement all of the sub-functions defined in the INT 15H, AX = 0D042H specification. There are no optional functions. BIOS must load the appropriate update for each processor during system initialization.

A Header Version of an update block containing the value 0FFFFFFFFH indicates that the update block is unused and available for storing a new update.

The BIOS is responsible for providing a region of non-volatile storage (NVRAM) for each potential processor stepping within a system. This storage unit consists of one or more update blocks. An update block is a contiguous 2048-byte block of memory. The BIOS for a single processor system need only provide update blocks to store one microcode update. If the BIOS for a multiple processor system is intended to support mixed processor steppings, then the BIOS needs to provide enough update blocks to store each unique microcode update or for each processor socket on the OEM's system board.

The BIOS is responsible for managing the NVRAM update blocks. This includes garbage collection, such as removing microcode updates that exist in NVRAM for which a corresponding processor does not exist in the system. This specification only provides the mechanism for ensuring security, the uniqueness of an entry, and that stale entries are not loaded. The actual update block management is implementation specific on a per-BIOS basis.

As an example, the BIOS may use update blocks sequentially in ascending order with CPU signatures sorted versus the first available block. In addition, garbage collection may be implemented as a setup option to clear all NVRAM slots or as BIOS code that searches and eliminates unused entries during boot.

NOTES

For IA-32 processors starting with family 0FH and model 03H and Intel 64 processors, the microcode update may be as large as 16 KBytes. Thus, BIOS must allocate 8 update blocks for each microcode update. In a MP system, a common microcode update may be sufficient for each socket in the system.

For IA-32 processors earlier than family 0FH and model 03H, the microcode update is 2 KBytes. An MP-capable BIOS that supports multiple steppings must allocate a block for each socket in the system.

A single-processor BIOS that supports variable-sized microcode update and fixed-sized microcode update must allocate one 16-KByte region and a second region of at least 2 KBytes.

The following algorithm (Example 12-11) describes the steps performed during BIOS initialization used to load the updates into the processor(s). The algorithm assumes:

- The BIOS ensures that no update contained within NVRAM has a header version or loader version that does not match one currently supported by the BIOS.
- The update contains a correct checksum.
- The BIOS ensures that (at most) one update exists for each processor stepping.
- Older update revisions are not allowed to overwrite more recent ones.

These requirements are checked by the BIOS during the execution of the write update function of this interface. The BIOS sequentially scans through all of the update blocks in NVRAM starting with index 0. The BIOS scans until it finds an update where the processor fields in the header match the processor signature (extended family, extended model, type, family, model, and stepping) as well as the platform bits of the current processor.

Example 12-11. Pseudo Code, Checks Required Prior to Loading an Update

```
For each processor in the system
{
```

```

Determine the Processor Signature via CPUID.01H;
Determine the Platform Bits  $\leftarrow 1 \ll \text{IA32\_PLATFORM\_ID}[52:50]$ ;

For (I  $\leftarrow$  UpdateBlock 0, I < NumOfBlocks; I++)
{
    If (Update.Header_Version = 00000001H)
    {
        If ((Update.ProcessorSignature = Processor Signature) &&
            (Update.ProcessorFlags & Platform Bits))
        {
            Load Update.UpdateData into the Processor;
            Verify update was correctly loaded into the processor
            Go on to next processor
            Break;
        }
        Else If (Update.TotalSize > (Update.DataSize + 48))
        {
            N  $\leftarrow$  0
            While (N < Update.ExtendedSignatureCount)
            {
                If ((Update.ProcessorSignature[N] =
                    Processor Signature) &&
                    (Update.ProcessorFlags[N] & Platform Bits))
                {
                    Load Update.UpdateData into the Processor;
                    Verify update correctly loaded into the processor
                    Go on to next processor
                    Break;
                }
                N  $\leftarrow$  N + 1
            }
            I  $\leftarrow$  I + (Update.TotalSize / 2048)
            If ((Update.TotalSize MOD 2048) = 0)
                I  $\leftarrow$  I + 1
        }
    }
}

```

NOTES

The platform Id bits in IA32_PLATFORM_ID are encoded as a three-bit binary coded decimal field. The platform bits in the microcode update header are individually bit encoded. The algorithm must do a translation from one format to the other prior to doing a check.

When performing the INT 15H, 0D042H functions, the BIOS must assume that the caller has no knowledge of platform specific requirements. It is the responsibility of BIOS calls to manage all chipset and platform specific prerequisites for managing the NVRAM device. When writing the update data using the Write Update sub-function, the BIOS must maintain implementation specific data requirements (such as the update of NVRAM checksum). The BIOS should also attempt to verify the success of write operations on the storage device used to record the update.

12.11.8.2 Responsibilities of the Calling Program

This section of the document lists the responsibilities of a calling program using the interface specifications to load microcode update(s) into BIOS NVRAM.

- The calling program should call the INT 15H, 0D042H functions from a pure real mode program and should be executing on a system that is running in pure real mode.

- The caller should issue the presence test function (sub function 0) and verify the signature and return codes of that function.
- It is important that the calling program provides the required scratch RAM buffers for the BIOS and the proper stack size as specified in the interface definition.
- The calling program should read any update data that already exists in the BIOS in order to make decisions about the appropriateness of loading the update. The BIOS must refuse to overwrite a newer update with an older version. The update header contains information about version and processor specifics for the calling program to make an intelligent decision about loading.
- There can be no ambiguous updates. The BIOS must refuse to allow multiple updates for the same CPU to exist at the same time; it also must refuse to load updates for processors that don't exist on the system.
- The calling application should implement a verify function that is run after the update write function successfully completes. This function reads back the update and verifies that the BIOS returned an image identical to the one that was written.

Example 12-12 represents a calling program.

Example 12-12. INT 15 D042 Calling Program Pseudo-code

```
//
// We must be in real mode
//
If the system is not in Real mode exit
//
// Detect presence of Genuine Intel processor(s) that can be updated
// using(CPUID)
//
If no Intel processors exist that can be updated exit
//
// Detect the presence of the Intel microcode update extensions
//
If the BIOS fails the PresenceTest exit
//
// If the APIC is enabled, see if any other processors are out there
//
Read IA32_APICBASE
If APIC enabled
{
    Send Broadcast Message to all processors except self via APIC
    Have all processors execute CPUID, record the Processor Signature
    (i.e., Extended Family, Extended Model, Type, Family, Model, Stepping)
    Have all processors read IA32_PLATFORM_ID[52:50], record Platform
    Id Bits

    If current processor cannot be updated
        exit
}
//
// Determine the number of unique update blocks needed for this system
//
NumBlocks = 0
For each processor
{
    If ((this is a unique processor stepping) AND
        (we have a unique update in the database for this processor))
    {
        Checksum the update from the database;
        If Checksum fails
            exit
    }
}
```

```

    NumBlocks ← NumBlocks + size of microcode update / 2048
}
}

//
// Do we have enough update slots for all CPUs?
//
If there are more blocks required to support the unique processor steppings than update blocks
provided by the BIOS exit
//
// Do we need any update blocks at all?  If not, we are done
//
If (NumBlocks = 0)
    exit
//
// Record updates for processors in NVRAM.
//
For (I=0; I<NumBlocks; I++)
{
    //
    // Load each Update
    //
    Issue the WriteUpdate function

    If (STORAGE_FULL) returned
    {
        Display Error -- BIOS is not managing NVRAM appropriately
        exit
    }

    If (INVALID_REVISION) returned
    {
        Display Message: More recent update already loaded in NVRAM for
        this stepping
        continue
    }

    If any other error returned
    {
        Display Diagnostic
        exit
    }

    //
    // Verify the update was loaded correctly
    //
    Issue the ReadUpdate function

    If an error occurred
    {
        Display Diagnostic
        exit
    }
    //
    // Compare the Update read to that written
    //
    If (Update read ≠ Update written)
    {
        Display Diagnostic
    }
}

```

```
        exit
    }

    I ← I + (size of microcode update / 2048)
}
//
// Enable Update Loading, and inform user
//
Issue the Update Control function with Task = Enable.
```

12.11.8.3 Microcode Update Functions

Table 12-13 defines the processor microcode update functions that implementations of INT 15H 0D042H must support.

Table 12-13. Microcode Update Functions

Microcode Update Function	Function Number	Description	Required/Optional
Presence test	00H	Returns information about the supported functions.	Required
Write update data	01H	Writes one of the update data areas (slots).	Required
Update control	02H	Globally controls the loading of updates.	Required
Read update data	03H	Reads one of the update data areas (slots).	Required

12.11.8.4 INT 15H-based Interface

If an OEM-BIOS is implementing INT 15H 0D042H interface and offer to its client, the BIOS should allow additional microcode updates to be added to system flash.

The program that calls this interface is responsible for providing three 64-kilobyte RAM areas for BIOS use during calls to the read and write functions. These RAM scratch pads can be used by the BIOS for any purpose, but only for the duration of the function call. The calling routine places real mode segments pointing to the RAM blocks in the CX, DX, and SI registers. Calls to functions in this interface must be made with a minimum of 32 kilobytes of stack available to the BIOS.

In general, each function returns with CF cleared and AH contains the returned status. The general return codes and other constant definitions are listed in Section 12.11.8.9, "Return Codes."

The OEM error field (AL) is provided for the OEM to return additional error information specific to the platform. If the BIOS provides no additional information about the error, OEM error must be set to SUCCESS. The OEM error field is undefined if AH contains either SUCCESS (00H) or NOT_IMPLEMENTED (86H). In all other cases, it must be set with either SUCCESS or a value meaningful to the OEM.

The following sections describe functions provided by the INT15H-based interface.

12.11.8.5 Function 00H—Presence Test

This function verifies that the BIOS has implemented required microcode update functions. Table 12-14 lists the parameters and return codes for the function.

Table 12-14. Parameters for the Presence Test

Input		
AX	Function Code	0D042H
BL	Sub-function	00H - Presence test
Output		
CF	Carry Flag	Carry Set - Failure - AH contains status Carry Clear - All return values valid

Table 12-14. Parameters for the Presence Test (Contd.)

Input		
AH	Return Code	
AL	OEM Error	Additional OEM information.
EBX	Signature Part 1	'INTE' - Part one of the signature
ECX	Signature Part 2	'LPEP' - Part two of the signature
EDX	Loader Version	Version number of the microcode update loader
SI	Update Count	Number of 2048 update blocks in NVRAM the BIOS allocated to storing microcode updates
Return Codes (see Table 12-19 for code definitions)		
SUCCESS		The function completed successfully.
NOT_IMPLEMENTED		The function is not implemented.

In order to assure that the BIOS function is present, the caller must verify the carry flag, the return code, and the 64-bit signature. The update count reflects the number of 2048-byte blocks available for storage within one non-volatile RAM.

The loader version number refers to the revision of the update loader program that is included in the system BIOS image.

12.11.8.6 Function 01H—Write Microcode Update Data

This function integrates a new microcode update into the BIOS storage device. Table 12-15 lists the parameters and return codes for the function.

Table 12-15. Parameters for the Write Update Data Function

Input		
AX	Function Code	0D042H
BL	Sub-function	01H - Write update
ES:DI	Update Address	Real Mode pointer to the Intel Update structure. This buffer is 2048 bytes in length if the processor supports only fixed-size microcode update or... Real Mode pointer to the Intel Update structure. This buffer is 64 KBytes in length if the processor supports a variable-size microcode update.
CX	Scratch Pad1	Real mode segment address of 64 KBytes of RAM block
DX	Scratch Pad2	Real mode segment address of 64 KBytes of RAM block
SI	Scratch Pad3	Real mode segment address of 64 KBytes of RAM block
SS:SP	Stack pointer	32 KBytes of stack minimum
Output		
CF	Carry Flag	Carry Set - Failure - AH Contains status Carry Clear - All return values valid
AH	Return Code	Status of the call
AL	OEM Error	Additional OEM information
Return Codes (see Table 12-19 for code definitions)		
SUCCESS		The function completed successfully.
NOT_IMPLEMENTED		The function is not implemented.

Table 12-15. Parameters for the Write Update Data Function (Contd.)

Input	
WRITE_FAILURE	A failure occurred because of the inability to write the storage device.
ERASE_FAILURE	A failure occurred because of the inability to erase the storage device.
READ_FAILURE	A failure occurred because of the inability to read the storage device.
STORAGE_FULL	The BIOS non-volatile storage area is unable to accommodate the update because all available update blocks are filled with updates that are needed for processors in the system.
CPU_NOT_PRESENT	The processor stepping does not currently exist in the system.
INVALID_HEADER	The update header contains a header or loader version that is not recognized by the BIOS.
INVALID_HEADER_CS	The update does not checksum correctly.
SECURITY_FAILURE	The processor rejected the update.
INVALID_REVISION	The same or more recent revision of the update exists in the storage device.

Description

The BIOS is responsible for selecting an appropriate update block in the non-volatile storage for storing the new update. This BIOS is also responsible for ensuring the integrity of the information provided by the caller, including authenticating the proposed update before incorporating it into storage.

Before writing the update block into NVRAM, the BIOS should ensure that the update structure meets the following criteria in the following order:

1. The update header version should be equal to an update header version recognized by the BIOS.
2. The update loader version in the update header should be equal to the update loader version contained within the BIOS image.
3. The update block must checksum. This checksum is computed as a 32-bit summation of all double words in the structure, including the header, data, and processor signature table.

The BIOS selects update block(s) in non-volatile storage for storing the candidate update. The BIOS can select any available update block as long as it guarantees that only a single update exists for any given processor stepping in non-volatile storage. If the update block selected already contains an update, the following additional criteria apply to overwrite it:

- The processor signature in the proposed update must be equal to the processor signature in the header of the current update in NVRAM (Processor Signature + platform ID bits).
- The update revision in the proposed update should be greater than the update revision in the header of the current update in NVRAM.

If no unused update blocks are available and the above criteria are not met, the BIOS can overwrite update block(s) for a processor stepping that is no longer present in the system. This can be done by scanning the update blocks and comparing the processor steppings, identified in the MP Specification table, to the processor steppings that currently exist in the system.

Finally, before storing the proposed update in NVRAM, the BIOS must verify the authenticity of the update via the mechanism described in Section 12.11.6, "Microcode Update Loader." This includes loading the update into the current processor, executing the CPUID instruction, reading MSR 08Bh, and comparing a calculated value with the update revision in the proposed update header for equality.

When performing the write update function, the BIOS must record the entire update, including the header, the update data, and the extended processor signature table (if applicable). When writing an update, the original contents may be overwritten, assuming the above criteria have been met. It is the responsibility of the BIOS to ensure that more recent updates are not overwritten through the use of this BIOS call, and that only a single update exists within the NVRAM for any processor stepping and platform ID.

Figure 12-8 and Figure 12-9 show the process the BIOS follows to choose an update block and ensure the integrity of the data when it stores the new microcode update.

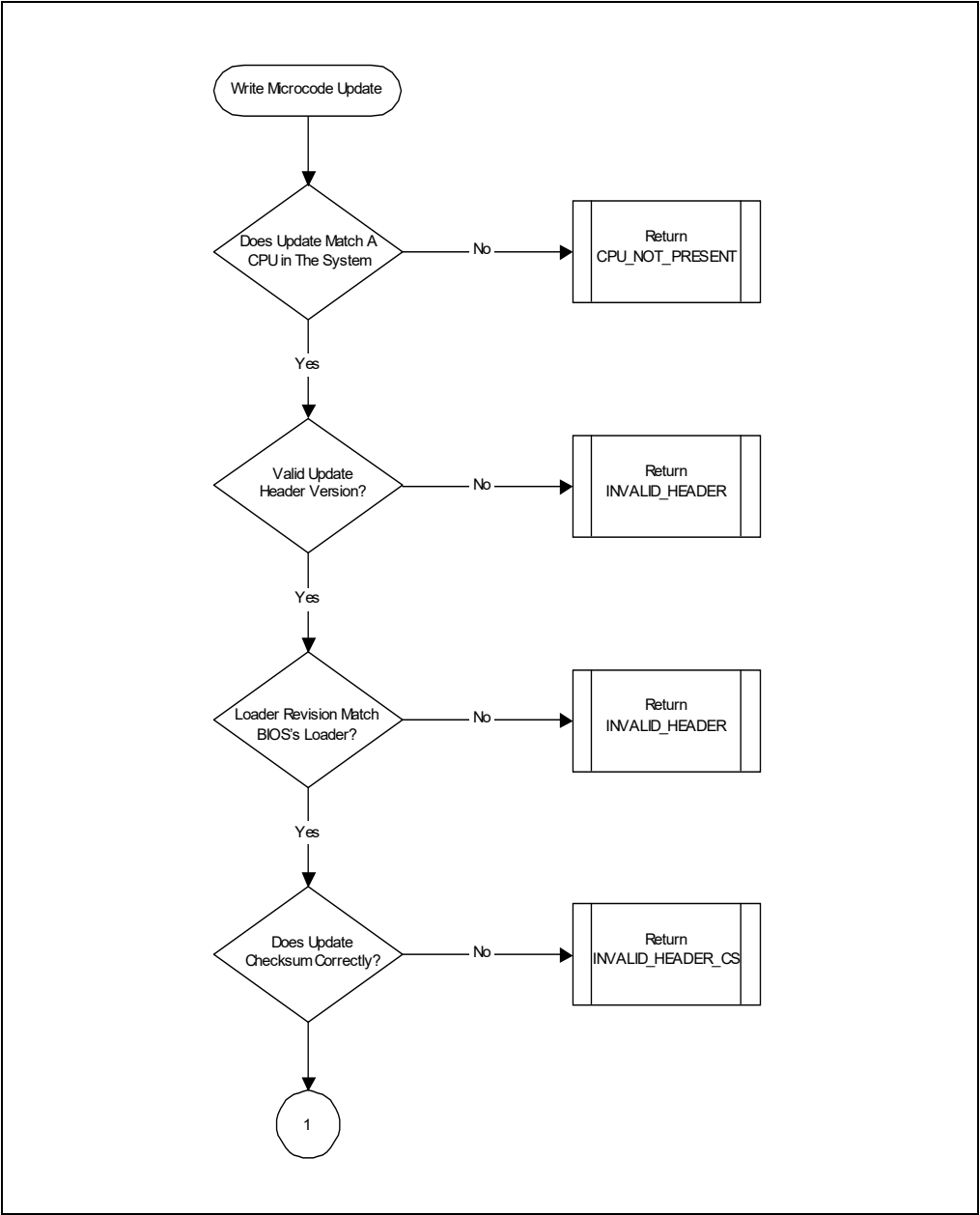


Figure 12-8. Microcode Update Write Operation Flow [1]

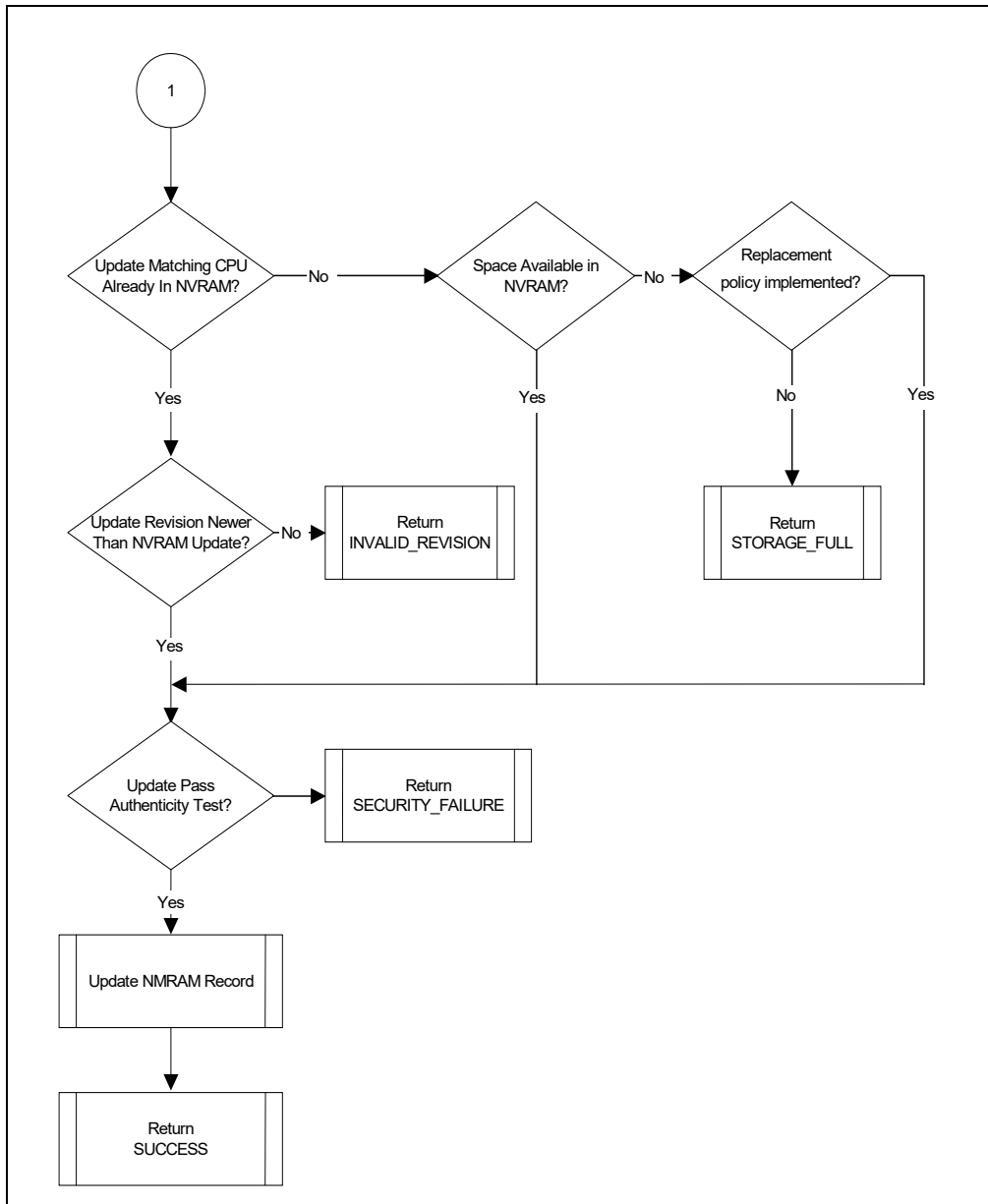


Figure 12-9. Microcode Update Write Operation Flow [2]

12.11.8.7 Function 02H—Microcode Update Control

This function enables loading of binary updates into the processor. Table 12-16 lists the parameters and return codes for the function.

Table 12-16. Parameters for the Control Update Sub-function

Input		
AX	Function Code	0D042H
BL	Sub-function	02H - Control update
BH	Task	See the description below.
CX	Scratch Pad1	Real mode segment of 64 KBytes of RAM block
DX	Scratch Pad2	Real mode segment of 64 KBytes of RAM block
SI	Scratch Pad3	Real mode segment of 64 KBytes of RAM block
SS:SP	Stack pointer	32 kilobytes of stack minimum
Output		
CF	Carry Flag	Carry Set - Failure - AH contains status Carry Clear - All return values valid.
AH	Return Code	Status of the call
AL	OEM Error	Additional OEM Information.
BL	Update Status	Either enable or disable indicator
Return Codes (see Table 12-19 for code definitions)		
SUCCESS		Function completed successfully.
READ_FAILURE		A failure occurred because of the inability to read the storage device.

This control is provided on a global basis for all updates and processors. The caller can determine the current status of update loading (enabled or disabled) without changing the state. The function does not allow the caller to disable loading of binary updates, as this poses a security risk.

The caller specifies the requested operation by placing one of the values from Table 12-17 in the BH register. After successfully completing this function, the BL register contains either the enable or the disable designator. Note that if the function fails, the update status return value is undefined.

Table 12-17. Mnemonic Values

Mnemonic	Value	Meaning
Enable	1	Enable the Update loading at initialization time.
Query	2	Determine the current state of the update control without changing its status.

The READ_FAILURE error code returned by this function has meaning only if the control function is implemented in the BIOS NVRAM. The state of this feature (enabled/disabled) can also be implemented using CMOS RAM bits where READ failure errors cannot occur.

12.11.8.8 Function 03H—Read Microcode Update Data

This function reads a currently installed microcode update from the BIOS storage into a caller-provided RAM buffer. Table 12-18 lists the parameters and return codes.

Table 12-18. Parameters for the Read Microcode Update Data Function

Input		
AX	Function Code	0D042H
BL	Sub-function	03H - Read Update
ES:DI	Buffer Address	Real Mode pointer to the Intel Update structure that will be written with the binary data

Table 12-18. Parameters for the Read Microcode Update Data Function (Contd.)

ECX	Scratch Pad1	Real Mode Segment address of 64 KBytes of RAM Block (lower 16 bits)
ECX	Scratch Pad2	Real Mode Segment address of 64 KBytes of RAM Block (upper 16 bits)
DX	Scratch Pad3	Real Mode Segment address of 64 KBytes of RAM Block
SS:SP	Stack pointer	32 KBytes of Stack Minimum
SI	Update Number	This is the index number of the update block to be read. This value is zero based and must be less than the update count returned from the presence test function.
Output		
CF	Carry Flag	Carry Set - Failure - AH contains Status
Carry Clear - All return values are valid.		
AH	Return Code	Status of the Call
AL	OEM Error	Additional OEM Information
Return Codes (see Table 12-19 for code definitions)		
SUCCESS		The function completed successfully.
READ_FAILURE		There was a failure because of the inability to read the storage device.
UPDATE_NUM_INVALID		Update number exceeds the maximum number of update blocks implemented by the BIOS.
NOT_EMPTY		The specified update block is a subsequent block in use to store a valid microcode update that spans multiple blocks. The specified block is not a header block and is not empty.

The read function enables the caller to read any microcode update data that already exists in a BIOS and make decisions about the addition of new updates. As a result of a successful call, the BIOS copies the microcode update into the location pointed to by ES:DI, with the contents of all Update block(s) that are used to store the specified microcode update.

If the specified block is not a header block, but does contain valid data from a microcode update that spans multiple update blocks, then the BIOS must return Failure with the NOT_EMPTY error code in AH.

An update block is considered unused and available for storing a new update if its Header Version contains the value 0FFFFFFFH after return from this function call. The actual implementation of NVRAM storage management is not specified here and is BIOS dependent. As an example, the actual data value used to represent an empty block by the BIOS may be zero, rather than 0FFFFFFFH. The BIOS is responsible for translating this information into the header provided by this function.

12.11.8.9 Return Codes

After the call has been made, the return codes listed in Table 12-19 are available in the AH register.

Table 12-19. Return Code Definitions

Return Code	Value	Description
SUCCESS	00H	The function completed successfully.
NOT_IMPLEMENTED	86H	The function is not implemented.
ERASE_FAILURE	90H	A failure because of the inability to erase the storage device.
WRITE_FAILURE	91H	A failure because of the inability to write the storage device.
READ_FAILURE	92H	A failure because of the inability to read the storage device.
STORAGE_FULL	93H	The BIOS non-volatile storage area is unable to accommodate the update because all available update blocks are filled with updates that are needed for processors in the system.
CPU_NOT_PRESENT	94H	The processor stepping does not currently exist in the system.
INVALID_HEADER	95H	The update header contains a header or loader version that is not recognized by the BIOS.
INVALID_HEADER_CS	96H	The update does not checksum correctly.
SECURITY_FAILURE	97H	The update was rejected by the processor.
INVALID_REVISION	98H	The same or more recent revision of the update exists in the storage device.
UPDATE_NUM_INVALID	99H	The update number exceeds the maximum number of update blocks implemented by the BIOS.
NOT_EMPTY	9AH	The specified update block is a subsequent block in use to store a valid microcode update that spans multiple blocks. The specified block is not a header block and is not empty.

18. Updates to Chapter 14, Volume 3A

Change bars and text show changes to Chapter 14 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- In Section 14.11.1, "MTRR Feature Identification", added PRMRR to the list of flags in the IA32_MTRRCAP MSR and to Figure 14-5.

This chapter describes the memory cache and cache control mechanisms, the TLBs, and the store buffer in Intel 64 and IA-32 processors. It also describes the memory type range registers (MTRRs) introduced in the P6 family processors and how they are used to control caching of physical memory locations.

14.1 INTERNAL CACHES, TLBS, AND BUFFERS

The Intel 64 and IA-32 architectures support cache, translation look aside buffers (TLBs), and a store buffer for temporary on-chip (and external) storage of instructions and data. (Figure 14-1 shows the arrangement of caches, TLBs, and the store buffer for the Pentium 4 and Intel Xeon processors.) Table 14-1 shows the characteristics of these caches and buffers for the Pentium 4, Intel Xeon, P6 family, and Pentium processors. **The sizes and characteristics of these units are machine specific and may change in future versions of the processor.** The CPUID instruction returns the sizes and characteristics of the caches and buffers for the processor on which the instruction is executed. See “CPUID—CPU Identification” in Chapter 3, “Instruction Set Reference, A-L,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.

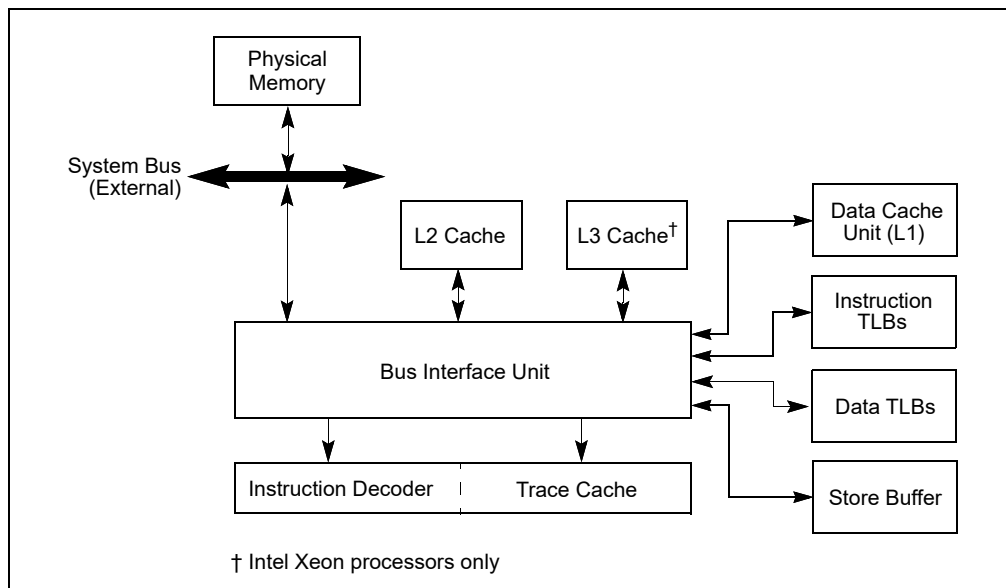


Figure 14-1. Cache Structure of the Pentium 4 and Intel Xeon Processors

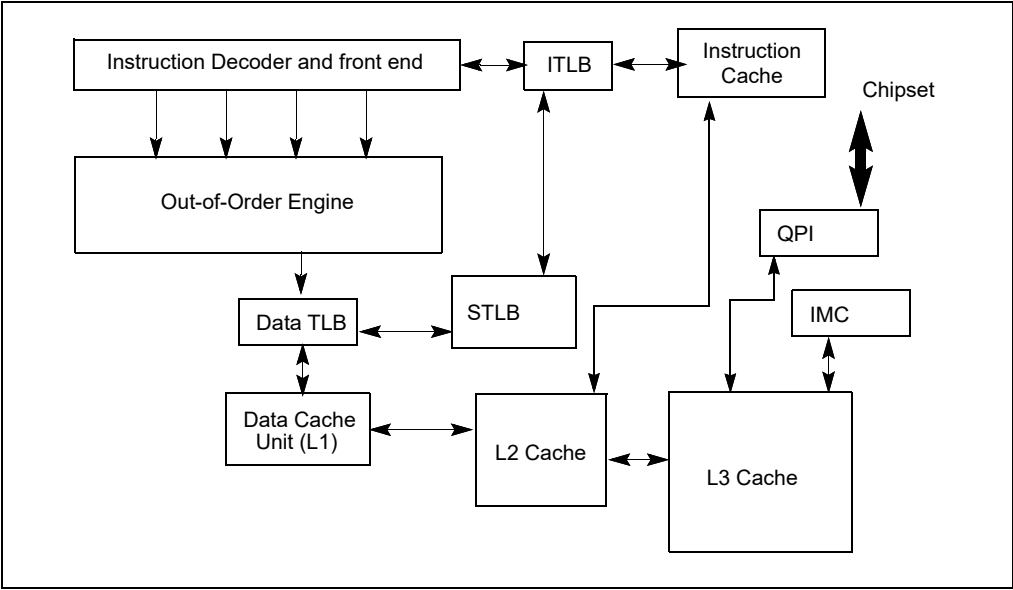


Figure 14-2. Cache Structure of the Intel Core i7 Processors

Figure 14-2 shows the cache arrangement of Intel Core i7 processor.

Table 14-1. Characteristics of the Caches, TLBs, Store Buffer, and Write Combining Buffer in Intel 64 and IA-32 Processors

Cache or Buffer	Characteristics
Trace Cache ¹	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst® microarchitecture): 12 Kμops, 8-way set associative. ▪ Intel Core i7, Intel Core 2 Duo, Intel Atom, Intel Core Duo, Intel Core Solo, Pentium M processor: not implemented. ▪ P6 family and Pentium processors: not implemented.
L1 Instruction Cache	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): not implemented. ▪ Intel Core i7 processor: 32-KByte, 4-way set associative. ▪ Intel Core 2 Duo, Intel Atom, Intel Core Duo, Intel Core Solo, Pentium M processor: 32-KByte, 8-way set associative. ▪ P6 family and Pentium processors: 8- or 16-KByte, 4-way set associative, 32-byte cache line size; 2-way set associative for earlier Pentium processors.
L1 Data Cache	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 8-KByte, 4-way set associative, 64-byte cache line size. ▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 16-KByte, 8-way set associative, 64-byte cache line size. ▪ Intel Atom processors: 24-KByte, 6-way set associative, 64-byte cache line size. ▪ Intel Core i7, Intel Core 2 Duo, Intel Core Duo, Intel Core Solo, Pentium M and Intel Xeon processors: 32-KByte, 8-way set associative, 64-byte cache line size. ▪ P6 family processors: 16-KByte, 4-way set associative, 32-byte cache line size; 8-KBytes, 2-way set associative for earlier P6 family processors. ▪ Pentium processors: 16-KByte, 4-way set associative, 32-byte cache line size; 8-KByte, 2-way set associative for earlier Pentium processors.

Table 14-1. Characteristics of the Caches, TLBs, Store Buffer, and Write Combining Buffer in Intel 64 and IA-32 Processors (Contd.)

Cache or Buffer	Characteristics
L2 Unified Cache	- Intel Core 2 Duo and Intel Xeon processors: up to 4-MByte (or 4MBx2 in quadcore processors), 16-way set associative, 64-byte cache line size. - Intel Core 2 Duo and Intel Xeon processors: up to 6-MByte (or 6MBx2 in quadcore processors), 24-way set associative, 64-byte cache line size. - Intel Core i7, i5, i3 processors: 256-KByte, 8-way set associative, 64-byte cache line size. - Intel Atom processors: 512-KByte, 8-way set associative, 64-byte cache line size. - Intel Core Duo, Intel Core Solo processors: 2-MByte, 8-way set associative, 64-byte cache line size - Pentium 4 and Intel Xeon processors: 256, 512, 1024, or 2048-KByte, 8-way set associative, 64-byte cache line size, 128-byte sector size. - Pentium M processor: 1 or 2-MByte, 8-way set associative, 64-byte cache line size. - P6 family processors: 128-KByte, 256-KByte, 512-KByte, 1-MByte, or 2-MByte, 4-way set associative, 32-byte cache line size. - Pentium processor (external optional): System specific, typically 256- or 512-KByte, 4-way set associative, 32-byte cache line size.
L3 Unified Cache	- Intel Xeon processors: 512-KByte, 1-MByte, 2-MByte, or 4-MByte, 8-way set associative, 64-byte cache line size, 128-byte sector size. - Intel Core i7 processor, Intel Xeon processor 5500: Up to 8MByte, 16-way set associative, 64-byte cache line size. - Intel Xeon processor 5600: Up to 12MByte, 64-byte cache line size. - Intel Xeon processor 7500: Up to 24MByte, 64-byte cache line size.
Instruction TLB (4-KByte Pages)	- Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 128 entries, 4-way set associative. - Intel Atom processors: 32-entries, fully associative. - Intel Core i7, i5, i3 processors: 64-entries per thread (128-entries per core), 4-way set associative. - Intel Core 2 Duo, Intel Core Duo, Intel Core Solo processors, Pentium M processor: 128 entries, 4-way set associative. - P6 family processors: 32 entries, 4-way set associative. - Pentium processor: 32 entries, 4-way set associative; fully set associative for Pentium processors with MMX technology.
Data TLB (4-KByte Pages)	- Intel Core i7, i5, i3 processors, DTLB0: 64-entries, 4-way set associative. - Intel Core 2 Duo processors: DTLB0, 16 entries, DTLB1, 256 entries, 4 ways. - Intel Atom processors: 16-entry-per-thread micro-TLB, fully associative; 64-entry DTLB, 4-way set associative; 16-entry PDE cache, fully associative. - Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 64 entry, fully set associative, shared with large page DTLB. - Intel Core Duo, Intel Core Solo processors, Pentium M processor: 128 entries, 4-way set associative. - Pentium and P6 family processors: 64 entries, 4-way set associative; fully set, associative for Pentium processors with MMX technology.
Instruction TLB (Large Pages)	- Intel Core i7, i5, i3 processors: 7-entries per thread, fully associative. - Intel Core 2 Duo processors: 4 entries, 4 ways. - Pentium 4 and Intel Xeon processors: large pages are fragmented. - Intel Core Duo, Intel Core Solo, Pentium M processor: 2 entries, fully associative. - P6 family processors: 2 entries, fully associative. - Pentium processor: Uses same TLB as used for 4-KByte pages.
Data TLB (Large Pages)	- Intel Core i7, i5, i3 processors, DTLB0: 32-entries, 4-way set associative. - Intel Core 2 Duo processors: DTLB0, 16 entries, DTLB1, 32 entries, 4 ways. - Intel Atom processors: 8 entries, 4-way set associative. - Pentium 4 and Intel Xeon processors: 64 entries, fully set associative; shared with small page data TLBs. - Intel Core Duo, Intel Core Solo, Pentium M processor: 8 entries, fully associative. - P6 family processors: 8 entries, 4-way set associative. - Pentium processor: 8 entries, 4-way set associative; uses same TLB as used for 4-KByte pages in Pentium processors with MMX technology.
Second-level Unified TLB (4-KByte Pages)	Intel Core i7, i5, i3 processor, STLB: 512-entries, 4-way set associative.

Table 14-1. Characteristics of the Caches, TLBs, Store Buffer, and Write Combining Buffer in Intel 64 and IA-32 Processors (Contd.)

Cache or Buffer	Characteristics
Store Buffer	- Intel Core i7, i5, i3 processors: 32 entries. - Intel Core 2 Duo processors: 20 entries. - Intel Atom processors: 8 entries, used for both WC and store buffers. - Pentium 4 and Intel Xeon processors: 24 entries. - Pentium M processor: 16 entries. - P6 family processors: 12 entries. - Pentium processor: 2 buffers, 1 entry each (Pentium processors with MMX technology have 4 buffers for 4 entries).
Write Combining (WC) Buffer	- Intel Core 2 Duo processors: 8 entries. - Intel Atom processors: 8 entries, used for both WC and store buffers. - Pentium 4 and Intel Xeon processors: 6 or 8 entries. - Intel Core Duo, Intel Core Solo, Pentium M processors: 6 entries. - P6 family processors: 4 entries.

NOTES:

1 Introduced to the IA-32 architecture in the Pentium 4 and Intel Xeon processors.

Intel 64 and IA-32 processors may implement four types of caches: the trace cache, the level 1 (L1) cache, the level 2 (L2) cache, and the level 3 (L3) cache. See Figure 14-1. Cache availability is described below:

- **Intel Core i7, i5, i3 processor family and Intel Xeon processor family based on Nehalem microarchitecture and Westmere microarchitecture** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache. Each processor core has its own L1 and L2. The L3 cache is an inclusive, unified data and instruction cache, shared by all processor cores inside a physical package. No trace cache is implemented.
- **Intel® Core™ 2 processor family and Intel® Xeon® processor family based on Intel® Core™ microarchitecture** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache located on the processor chip; it is shared between two processor cores in a dual-core processor implementation. Quad-core processors have two L2, each shared by two processor cores. No trace cache is implemented.
- **Intel Atom® processor** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache is located on the processor chip. No trace cache is implemented.
- **Intel® Core™ Solo and Intel® Core™ Duo processors** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache located on the processor chip. It is shared between two processor cores in a dual-core processor implementation. No trace cache is implemented.
- **Pentium® 4 and Intel® Xeon® processors Based on Intel NetBurst® microarchitecture** — The trace cache caches decoded instructions (μ ops) from the instruction decoder and the L1 cache contains data. The L2 and L3 caches are unified data and instruction caches located on the processor chip. Dualcore processors have two L2, one in each processor core. Note that the L3 cache is only implemented on some Intel Xeon processors.
- **P6 family processors** — The L1 cache is divided into two sections: one dedicated to caching instructions (pre-decoded instructions) and the other to caching data. The L2 cache is a unified data and instruction cache located on the processor chip. P6 family processors do not implement a trace cache.
- **Pentium® processors** — The L1 cache has the same structure as on P6 family processors. There is no trace cache. The L2 cache is a unified data and instruction cache external to the processor chip on earlier Pentium processors and implemented on the processor chip in later Pentium processors. For Pentium processors where the L2 cache is external to the processor, access to the cache is through the system bus.

For Intel Core i7 processors and processors based on Intel Core, Intel Atom, and Intel NetBurst microarchitectures, Intel Core Duo, Intel Core Solo and Pentium M processors, the cache lines for the L1 and L2 caches (and L3 caches if supported) are 64 bytes wide. The processor always reads a cache line from system memory beginning on a 64-byte boundary. (A 64-byte aligned cache line begins at an address with its 6 least-significant bits clear.) A cache

line can be filled from memory with a 8-transfer burst transaction. The caches do not support partially-filled cache lines, so caching even a single doubleword requires caching an entire line.

The L1 and L2 cache lines in the P6 family and Pentium processors are 32 bytes wide, with cache line reads from system memory beginning on a 32-byte boundary (5 least-significant bits of a memory address clear.) A cache line can be filled from memory with a 4-transfer burst transaction. Partially-filled cache lines are not supported.

The trace cache in processors based on Intel NetBurst microarchitecture is available in all execution modes: protected mode, system management mode (SMM), and real-address mode. The L1, L2, and L3 caches are also available in all execution modes; however, use of them must be handled carefully in SMM (see Section 34.4.2, “SMRAM Caching”).

The TLBs store the most recently used page-directory and page-table entries. They speed up memory accesses when paging is enabled by reducing the number of memory accesses that are required to read the page tables stored in system memory. The TLBs are divided into four groups: instruction TLBs for 4-KByte pages, data TLBs for 4-KByte pages; instruction TLBs for large pages (2-MByte, 4-MByte or 1-GByte pages), and data TLBs for large pages. The TLBs are normally active only in protected mode with paging enabled. When paging is disabled or the processor is in real-address mode, the TLBs maintain their contents until explicitly or implicitly flushed (see Section 14.9, “Invalidating the Translation Lookaside Buffers (TLBs)”).

Processors based on Intel Core microarchitectures implement one level of instruction TLB and two levels of data TLB. Intel Core i7 processor provides a second-level unified TLB.

The store buffer is associated with the processors instruction execution units. It allows writes to system memory and/or the internal caches to be saved and in some cases combined to optimize the processor’s bus accesses. The store buffer is always enabled in all execution modes.

The processor’s caches are for the most part transparent to software. When enabled, instructions and data flow through these caches without the need for explicit software control. However, knowledge of the behavior of these caches may be useful in optimizing software performance. For example, knowledge of cache dimensions and replacement algorithms gives an indication of how large of a data structure can be operated on at once without causing cache thrashing.

In multiprocessor systems, maintenance of cache consistency may, in rare circumstances, require intervention by system software. For these rare cases, the processor provides privileged cache control instructions for use in flushing caches and forcing memory ordering.

There are several instructions that software can use to improve the performance of the L1, L2, and L3 caches, including the PREFETCHh, CLFLUSH, and CLFLUSHOPT instructions and the non-temporal move instructions (MOVNTI, MOVNTDQ, MOVNTPS, and MOVNTPD). The use of these instructions are discussed in Section 14.5.5, “Cache Management Instructions.”

14.2 CACHING TERMINOLOGY

IA-32 processors (beginning with the Pentium processor) and Intel 64 processors use the MESI (modified, exclusive, shared, invalid) cache protocol to maintain consistency with internal caches and caches in other processors (see Section 14.4, “Cache Control Protocol”).

When the processor recognizes that an operand being read from memory is cacheable, the processor reads an entire cache line into the appropriate cache (L1, L2, L3, or all). This operation is called a **cache line fill**. If the memory location containing that operand is still cached the next time the processor attempts to access the operand, the processor can read the operand from the cache instead of going back to memory. This operation is called a **cache hit**.

When the processor attempts to write an operand to a cacheable area of memory, it first checks if a cache line for that memory location exists in the cache. If a valid cache line does exist, the processor (depending on the write policy currently in force) can write the operand into the cache instead of writing it out to system memory. This operation is called a **write hit**. If a write misses the cache (that is, a valid cache line is not present for area of memory being written to), the processor performs a cache line fill, write allocation. Then it writes the operand into the cache line and (depending on the write policy currently in force) can also write it out to memory. If the operand is to be written out to memory, it is written first into the store buffer, and then written from the store buffer to memory when the system bus is available. (Note that for the Pentium processor, write misses do not result in a cache line fill; they always result in a write to memory. For this processor, only read misses result in cache line fills.)

When operating in an MP system, IA-32 processors (beginning with the Intel486 processor) and Intel 64 processors have the ability to **snoop** other processor's accesses to system memory and to their internal caches. They use this snooping ability to keep their internal caches consistent both with system memory and with the caches in other processors on the bus. For example, in the Pentium and P6 family processors, if through snooping one processor detects that another processor intends to write to a memory location that it currently has cached in **shared state**, the snooping processor will invalidate its cache line forcing it to perform a cache line fill the next time it accesses the same memory location.

Beginning with the P6 family processors, if a processor detects (through snooping) that another processor is trying to access a memory location that it has modified in its cache, but has not yet written back to system memory, the snooping processor will signal the other processor (by means of the HITM# signal) that the cache line is held in modified state and will perform an implicit write-back of the modified data. The implicit write-back is transferred directly to the initial requesting processor and snooped by the memory controller to assure that system memory has been updated. Here, the processor with the valid data may pass the data to the other processors without actually writing it to system memory; however, it is the responsibility of the memory controller to snoop this operation and update memory.

14.3 METHODS OF CACHING AVAILABLE

The processor allows any area of system memory to be cached in the L1, L2, and L3 caches. In individual pages or regions of system memory, it allows the type of caching (also called **memory type**) to be specified (see Section 14.5). Memory types currently defined for the Intel 64 and IA-32 architectures are (see Table 14-2):

- **Strong Uncacheable (UC)** —System memory locations are not cached. All reads and writes appear on the system bus and are executed in program order without reordering. No speculative memory accesses, page-table walks, or prefetches of speculated branch targets are made. This type of cache-control is useful for memory-mapped I/O devices. When used with normal RAM, it greatly reduces processor performance.

NOTE

The behavior of x87 and SIMD instructions referencing memory is implementation dependent. In some implementations, accesses to UC memory may occur more than once. To ensure predictable behavior, use loads and stores of general purpose registers to access UC memory that may have read or write side effects.

Table 14-2. Memory Types and Their Properties

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Strong Uncacheable (UC)	No	No	No	Strong Ordering
Uncacheable (UC-)	No	No	No	Strong Ordering. Can only be selected through the PAT. Can be overridden by WC in MTRRs.
Write Combining (WC)	No	No	Yes	Weak Ordering. Available by programming MTRRs or by selecting it through the PAT.
Write Through (WT)	Yes	No	Yes	Speculative Processor Ordering.
Write Back (WB)	Yes	Yes	Yes	Speculative Processor Ordering.
Write Protected (WP)	Yes for reads; no for writes	No	Yes	Speculative Processor Ordering. Available by programming MTRRs.

- **Uncacheable (UC-)** — Has same characteristics as the strong uncacheable (UC) memory type, except that this memory type can be overridden by programming the MTRRs for the WC memory type. This memory type is available in processor families starting from the Pentium III processors and can only be selected through the PAT.

- **Write Combining (WC)** — System memory locations are not cached (as with uncacheable memory) and coherency is not enforced by the processor's bus coherency protocol. Speculative reads are allowed. Writes may be delayed and combined in the write combining buffer (WC buffer) to reduce memory accesses. If the WC buffer is partially filled, the writes may be delayed until the next occurrence of a serializing event; such as an SFENCE or MFENCE instruction, CPUID or other serializing instruction, a read or write to uncached memory, an interrupt occurrence, or an execution of a LOCK instruction (including one with an XACQUIRE or XRELEASE prefix). In addition, an execution of the XEND instruction (to end a transactional region) evicts any writes that were buffered before the corresponding execution of the XBEGIN instruction (to begin the transactional region) before evicting any writes that were performed inside the transactional region.

This type of cache-control is appropriate for video frame buffers, where the order of writes is unimportant as long as the writes update memory so they can be seen on the graphics display. See Section 14.3.1, "Buffering of Write Combining Memory Locations," for more information about caching the WC memory type. This memory type is available in the Pentium Pro and Pentium II processors by programming the MTRRs; or in processor families starting from the Pentium III processors by programming the MTRRs or by selecting it through the PAT.

- **Write-through (WT)** — Writes and reads to and from system memory are cached. Reads come from cache lines on cache hits; read misses cause cache fills. Speculative reads are allowed. All writes are written to a cache line (when possible) and through to system memory. When writing through to memory, invalid cache lines are never filled, and valid cache lines are either filled or invalidated. Write combining is allowed. This type of cache-control is appropriate for frame buffers or when there are devices on the system bus that access system memory, but do not perform snooping of memory accesses. It enforces coherency between caches in the processors and system memory.
- **Write-back (WB)** — Writes and reads to and from system memory are cached. Reads come from cache lines on cache hits; read misses cause cache fills. Speculative reads are allowed. Write misses cause cache line fills (in processor families starting with the P6 family processors), and writes are performed entirely in the cache, when possible. Write combining is allowed. The write-back memory type reduces bus traffic by eliminating many unnecessary writes to system memory. Writes to a cache line are not immediately forwarded to system memory; instead, they are accumulated in the cache. The modified cache lines are written to system memory later, when a write-back operation is performed. Write-back operations are triggered when cache lines need to be deallocated, such as when new cache lines are being allocated in a cache that is already full. They also are triggered by the mechanisms used to maintain cache consistency. This type of cache-control provides the best performance, but it requires that all devices that access system memory on the system bus be able to snoop memory accesses to ensure system memory and cache coherency.
- **Write protected (WP)** — Reads come from cache lines when possible, and read misses cause cache fills. Writes are propagated to the system bus and cause corresponding cache lines on all processors on the bus to be invalidated. Speculative reads are allowed. This memory type is available in processor families starting from the P6 family processors by programming the MTRRs (see Table 14-6).

Table 14-3 shows which of these caching methods are available in the Pentium, P6 Family, Pentium 4, and Intel Xeon processors.

Table 14-3. Methods of Caching Available in Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium M, Pentium 4, Intel Xeon, P6 Family, and Pentium Processors

Memory Type	Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium M, Pentium 4 and Intel Xeon Processors	P6 Family Processors	Pentium Processor
Strong Uncacheable (UC)	Yes	Yes	Yes
Uncacheable (UC-)	Yes	Yes*	No
Write Combining (WC)	Yes	Yes	No
Write Through (WT)	Yes	Yes	Yes
Write Back (WB)	Yes	Yes	Yes
Write Protected (WP)	Yes	Yes	No

NOTE:

* Introduced in the Pentium III processor; not available in the Pentium Pro or Pentium II processors

14.3.1 Buffering of Write Combining Memory Locations

Writes to the WC memory type are not cached in the typical sense of the word cached. They are retained in an internal write combining buffer (WC buffer) that is separate from the internal L1, L2, and L3 caches and the store buffer. The WC buffer is not snooped and thus does not provide data coherency. Buffering of writes to WC memory is done to allow software a small window of time to supply more modified data to the WC buffer while remaining as non-intrusive to software as possible. The buffering of writes to WC memory also causes data to be collapsed; that is, multiple writes to the same memory location will leave the last data written in the location and the other writes will be lost.

The size and structure of the WC buffer is not architecturally defined. For the Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium M, Pentium 4 and Intel Xeon processors; the WC buffer is made up of several 64-byte WC buffers. For the P6 family processors, the WC buffer is made up of several 32-byte WC buffers.

When software begins writing to WC memory, the processor begins filling the WC buffers one at a time. When one or more WC buffers has been filled, the processor has the option of evicting the buffers to system memory. The protocol for evicting the WC buffers is implementation dependent and should not be relied on by software for system memory coherency. When using the WC memory type, software **must** be sensitive to the fact that the writing of data to system memory is being delayed and **must** deliberately empty the WC buffers when system memory coherency is required.

Once the processor has started to evict data from the WC buffer into system memory, it will make a bus-transaction style decision based on how much of the buffer contains valid data. If the buffer is full (for example, all bytes are valid), the processor will execute a burst-write transaction on the bus. This results in all 32 bytes (P6 family processors) or 64 bytes (Pentium 4 and more recent processor) being transmitted on the data bus in a single burst transaction. If one or more of the WC buffer's bytes are invalid (for example, have not been written by software), the processor will transmit the data to memory using "partial write" transactions (one chunk at a time, where a "chunk" is 8 bytes).

This will result in a maximum of 4 partial write transactions (for P6 family processors) or 8 partial write transactions (for the Pentium 4 and more recent processors) for one WC buffer of data sent to memory.

The WC memory type is weakly ordered by definition. Once the eviction of a WC buffer has started, the data is subject to the weak ordering semantics of its definition. Ordering is not maintained between the successive allocation/deallocation of WC buffers (for example, writes to WC buffer 1 followed by writes to WC buffer 2 may appear as buffer 2 followed by buffer 1 on the system bus). When a WC buffer is evicted to memory as partial writes there is no guaranteed ordering between successive partial writes (for example, a partial write for chunk 2 may appear on the bus before the partial write for chunk 1 or vice versa).

The only elements of WC propagation to the system bus that are guaranteed are those provided by transaction atomicity. For example, with a P6 family processor, a completely full WC buffer will always be propagated as a single 32-bit burst transaction using any chunk order. In a WC buffer eviction where data will be evicted as partials, all data contained in the same chunk (0 mod 8 aligned) will be propagated simultaneously. Likewise, for more recent processors starting with those based on Intel NetBurst microarchitectures, a full WC buffer will always be propagated as a single burst transactions, using any chunk order within a transaction. For partial buffer propagations, all data contained in the same chunk will be propagated simultaneously.

14.3.2 Choosing a Memory Type

The simplest system memory model does not use memory-mapped I/O with read or write side effects, does not include a frame buffer, and uses the write-back memory type for all memory. An I/O agent can perform direct memory access (DMA) to write-back memory and the cache protocol maintains cache coherency.

A system can use strong uncacheable memory for other memory-mapped I/O, and should always use strong uncacheable memory for memory-mapped I/O with read side effects.

Dual-ported memory can be considered a write side effect, making relatively prompt writes desirable, because those writes cannot be observed at the other port until they reach the memory agent. A system can use strong uncacheable, uncacheable, write-through, or write-combining memory for frame buffers or dual-ported memory that contains pixel values displayed on a screen. Frame buffer memory is typically large (a few megabytes) and is usually written more than it is read by the processor. Using strong uncacheable memory for a frame buffer generates very large amounts of bus traffic, because operations on the entire buffer are implemented using partial writes rather than line writes. Using write-through memory for a frame buffer can displace almost all other useful cached

lines in the processor's L2 and L3 caches and L1 data cache. Therefore, systems should use write-combining memory for frame buffers whenever possible.

Software can use page-level cache control, to assign appropriate effective memory types when software will not access data structures in ways that benefit from write-back caching. For example, software may read a large data structure once and not access the structure again until the structure is rewritten by another agent. Such a large data structure should be marked as uncacheable, or reading it will evict cached lines that the processor will be referencing again.

A similar example would be a write-only data structure that is written to (to export the data to another agent), but never read by software. Such a structure can be marked as uncacheable, because software never reads the values that it writes (though as uncacheable memory, it will be written using partial writes, while as write-back memory, it will be written using line writes, which may not occur until the other agent reads the structure and triggers implicit write-backs).

On the Pentium III, Pentium 4, and more recent processors, new instructions are provided that give software greater control over the caching, prefetching, and the write-back characteristics of data. These instructions allow software to use weakly ordered or processor ordered memory types to improve processor performance, but when necessary to force strong ordering on memory reads and/or writes. They also allow software greater control over the caching of data. For a description of these instructions and their intended use, see Section 14.5.5, "Cache Management Instructions."

14.3.3 Code Fetches in Uncacheable Memory

Programs may execute code from uncacheable (UC) memory, but the implications are different from accessing data in UC memory. When doing code fetches, the processor never transitions from cacheable code to UC code speculatively. It also never speculatively fetches branch targets that result in UC code.

The processor may fetch the same UC cache line multiple times in order to decode an instruction once. It may decode consecutive UC instructions in a cache line without fetching between each instruction. It may also fetch additional cachelines from the same or a consecutive 4-KByte page in order to decode one non-speculative UC instruction (this can be true even when the instruction is contained fully in one line).

Because of the above and because cache line sizes may change in future processors, software should avoid placing memory-mapped I/O with read side effects in the same page or in a subsequent page used to execute UC code.

14.4 CACHE CONTROL PROTOCOL

The following section describes the cache control protocol currently defined for the Intel 64 and IA-32 architectures.

In the L1 data cache and in the L2/L3 unified caches, the MESI (modified, exclusive, shared, invalid) cache protocol maintains consistency with caches of other processors. The L1 data cache and the L2/L3 unified caches have two MESI status flags per cache line. Each line can be marked as being in one of the states defined in Table 14-4. In general, the operation of the MESI protocol is transparent to programs.

Table 14-4. MESI Cache Line States

Cache Line State	M (Modified)	E (Exclusive)	S (Shared)	I (Invalid)
This cache line is valid?	Yes	Yes	Yes	No
The memory copy is...	Out of date	Valid	Valid	—
Copies exist in caches of other processors?	No	No	Maybe	Maybe
A write to this line ...	Does not go to the system bus.	Does not go to the system bus.	Causes the processor to gain exclusive ownership of the line.	Goes directly to the system bus.

The L1 instruction cache in P6 family processors implements only the “SI” part of the MESI protocol, because the instruction cache is not writable. The instruction cache monitors changes in the data cache to maintain consistency between the caches when instructions are modified. See Section 14.6, “Self-Modifying Code,” for more information on the implications of caching instructions.

14.5 CACHE CONTROL

The Intel 64 and IA-32 architectures provide a variety of mechanisms for controlling the caching of data and instructions and for controlling the ordering of reads and writes between the processor, the caches, and memory. These mechanisms can be divided into two groups:

- **Cache control registers and bits** — The Intel 64 and IA-32 architectures define several dedicated registers and various bits within control registers and page- and directory-table entries that control the caching system memory locations in the L1, L2, and L3 caches. These mechanisms control the caching of virtual memory pages and of regions of physical memory.
- **Cache control and memory ordering instructions** — The Intel 64 and IA-32 architectures provide several instructions that control the caching of data, the ordering of memory reads and writes, and the prefetching of data. These instructions allow software to control the caching of specific data structures, to control memory coherency for specific locations in memory, and to force strong memory ordering at specific locations in a program.

The following sections describe these two groups of cache control mechanisms.

14.5.1 Cache Control Registers and Bits

Figure 14-3 depicts cache-control mechanisms in IA-32 processors. Other than for the matter of memory address space, these work the same in Intel 64 processors.

The Intel 64 and IA-32 architectures provide the following cache-control registers and bits for use in enabling or restricting caching to various pages or regions in memory:

- **CD flag, bit 30 of control register CR0** — Controls caching of system memory locations (see Section 2.5, “Control Registers”). If the CD flag is clear, caching is enabled for the whole of system memory, but may be restricted for individual pages or regions of memory by other cache-control mechanisms. When the CD flag is set, caching is restricted in the processor’s caches (cache hierarchy) for the P6 and more recent processor families and prevented for the Pentium processor (see note below). With the CD flag set, however, the caches will still respond to snoop traffic. Caches should be explicitly flushed to ensure memory coherency. For highest processor performance, both the CD and the NW flags in control register CR0 should be cleared. Table 14-5 shows the interaction of the CD and NW flags.

The effect of setting the CD flag is somewhat different for processor families starting with P6 family than the Pentium processor (see Table 14-5). To ensure memory coherency after the CD flag is set, the caches should be explicitly flushed (see Section 14.5.3, “Preventing Caching”). Setting the CD flag for the P6 and more recent processor families modifies cache line fill and update behavior. Also, setting the CD flag on these processors do not force strict ordering of memory accesses unless the MTRRs are disabled and/or all memory is referenced as uncached (see Section 11.2.5, “Strengthening or Weakening the Memory-Ordering Model”).

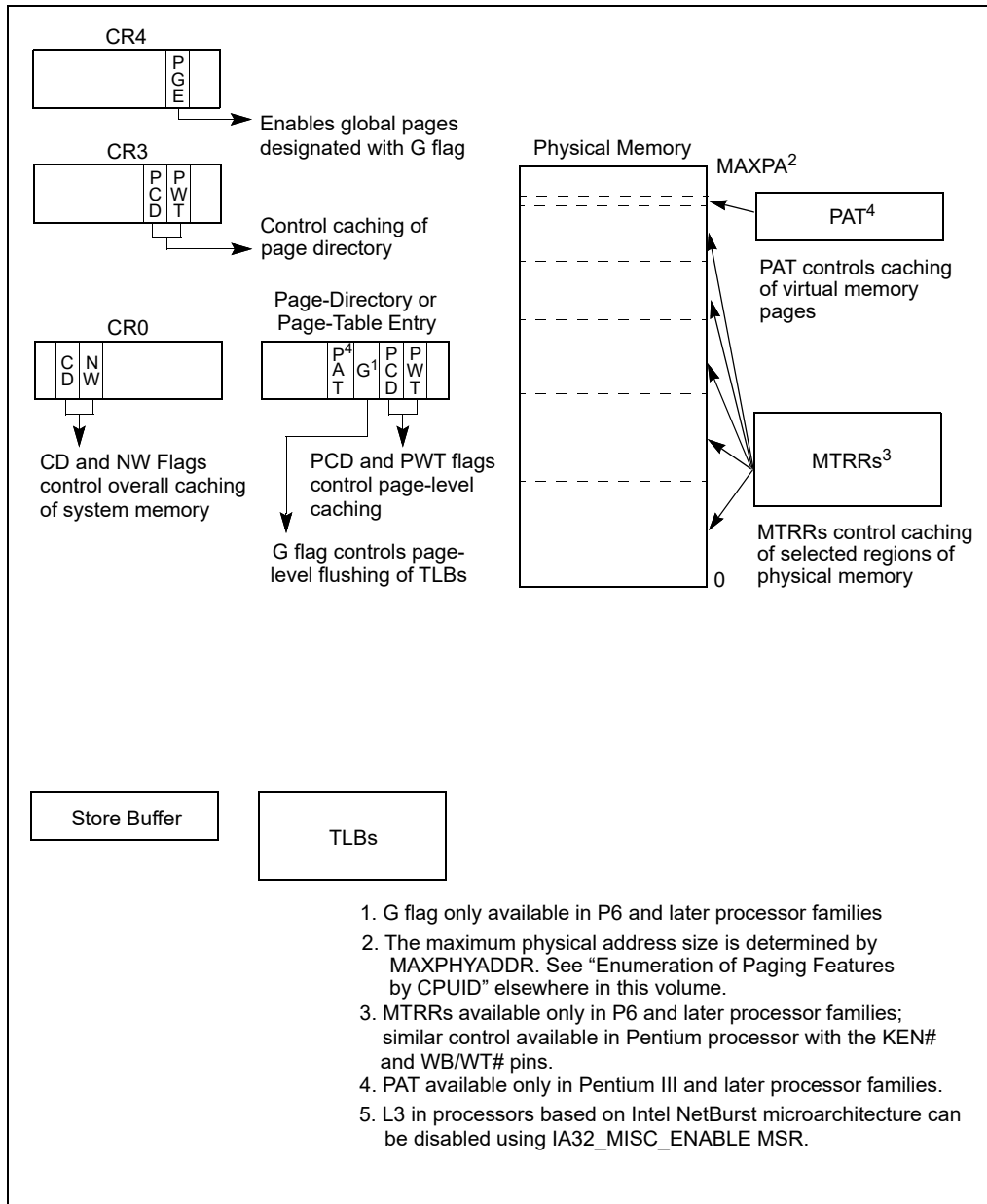


Figure 14-3. Cache-Control Registers and Bits Available in Intel 64 and IA-32 Processors

Table 14-5. Cache Operating Modes

CD	NW	Caching and Read/Write Policy	L1	L2/L3 ¹
0	0	Normal Cache Mode. Highest performance cache operation. - Read hits access the cache; read misses may cause replacement. - Write hits update the cache. - Only writes to shared lines and write misses update system memory. - Write misses cause cache line fills. - Write hits can change shared lines to modified under control of the MTRRs and with associated read invalidation cycle. (Pentium processor only.) Write misses do not cause cache line fills. (Pentium processor only.) Write hits can change shared lines to exclusive under control of WB/WT#. - Invalidation is allowed. - External snoop traffic is supported.	Yes Yes Yes Yes Yes Yes Yes Yes	Yes Yes Yes Yes Yes Yes
0	1	Invalid setting. Generates a general-protection exception (#GP) with an error code of 0.	NA	NA
1	0	No-fill Cache Mode. Memory coherency is maintained.³ (Pentium 4 and later processor families.) State of processor after a power up or reset. - Read hits access the cache; read misses do not cause replacement (see Pentium 4 and Intel Xeon processors reference below). - Write hits update the cache. - Only writes to shared lines and write misses update system memory. - Write misses access memory. - Write hits can change shared lines to exclusive under control of the MTRRs and with associated read invalidation cycle. (Pentium processor only.) Write hits can change shared lines to exclusive under control of the WB/WT#. (P6 and later processor families only.) Strict memory ordering is not enforced unless the MTRRs are disabled and/or all memory is referenced as uncached (see Section 7.2.4., "Strengthening or Weakening the Memory Ordering Model"). - Invalidation is allowed. - External snoop traffic is supported.	Yes Yes Yes Yes Yes Yes Yes Yes Yes	Yes Yes Yes Yes Yes Yes Yes Yes
1	1	Memory coherency is not maintained.^{2, 3} (P6 family and Pentium processors.) State of the processor after a power up or reset. - Read hits access the cache; read misses do not cause replacement. - Write hits update the cache and change exclusive lines to modified. - Shared lines remain shared after write hit. - Write misses access memory. - Invalidation is inhibited when snooping; but is allowed with INVD and WBINVD instructions. - External snoop traffic is supported.	Yes Yes Yes Yes Yes Yes Yes No	Yes Yes Yes Yes Yes Yes Yes Yes

NOTES:

1. The L2/L3 column in this table is definitive for the Pentium 4, Intel Xeon, and P6 family processors. It is intended to represent what could be implemented in a system based on a Pentium processor with an external, platform specific, write-back L2 cache.
2. The Pentium 4 and more recent processor families do not support this mode; setting the CD and NW bits to 1 selects the no-fill cache mode.
3. Not supported In Intel Atom processors. If CD = 1 in an Intel Atom processor, caching is disabled.

- **NW flag, bit 29 of control register CR0** — Controls the write policy for system memory locations (see Section 2.5, “Control Registers”). If the NW and CD flags are clear, write-back is enabled for the whole of system memory, but may be restricted for individual pages or regions of memory by other cache-control mechanisms. Table 14-5 shows how the other combinations of CD and NW flags affects caching.

NOTES

For the Pentium 4 and Intel Xeon processors, the NW flag is a don't care flag; that is, when the CD flag is set, the processor uses the no-fill cache mode, regardless of the setting of the NW flag.

For Intel Atom processors, the NW flag is a don't care flag; that is, when the CD flag is set, the processor disables caching, regardless of the setting of the NW flag.

For the Pentium processor, when the L1 cache is disabled (the CD and NW flags in control register CR0 are set), external snoops are accepted in DP (dual-processor) systems and inhibited in uniprocessor systems.

When snoops are inhibited, address parity is not checked and APCHK# is not asserted for a corrupt address; however, when snoops are accepted, address parity is checked and APCHK# is asserted for corrupt addresses.

- **PCD and PWT flags in paging-structure entries** — Control the memory type used to access paging structures and pages (see Section 5.9, “Paging and Memory Typing”).
- **PCD and PWT flags in control register CR3** — Control the memory type used to access the first paging structure of the current paging-structure hierarchy (see Section 5.9, “Paging and Memory Typing”).
- **G (global) flag in the page-directory and page-table entries (introduced to the IA-32 architecture in the P6 family processors)** — Controls the flushing of TLB entries for individual pages. See Section 5.10, “Caching Translation Information,” for more information about this flag.
- **PGE (page global enable) flag in control register CR4** — Enables the establishment of global pages with the G flag. See Section 5.10, “Caching Translation Information,” for more information about this flag.
- **Memory type range registers (MTRRs) (introduced in P6 family processors)** — Control the type of caching used in specific regions of physical memory. Any of the caching types described in Section 14.3, “Methods of Caching Available,” can be selected. See Section 14.11, “Memory Type Range Registers (MTRRs),” for a detailed description of the MTRRs.
- **Page Attribute Table (PAT) MSR (introduced in the Pentium III processor)** — Extends the memory typing capabilities of the processor to permit memory types to be assigned on a page-by-page basis (see Section 14.12, “Page Attribute Table (PAT)”).
- **Third-Level Cache Disable flag, bit 6 of the IA32_MISC_ENABLE MSR (Available only in processors based on Intel NetBurst microarchitecture)** — Allows the L3 cache to be disabled and enabled, independently of the L1 and L2 caches.
- **KEN# and WB/WT# pins (Pentium processor)** — Allow external hardware to control the caching method used for specific areas of memory. They perform similar (but not identical) functions to the MTRRs in the P6 family processors.
- **PCD and PWT pins (Pentium processor)** — These pins (which are associated with the PCD and PWT flags in control register CR3 and in the page-directory and page-table entries) permit caching in an external L2 cache to be controlled on a page-by-page basis, consistent with the control exercised on the L1 cache of these processors. The P6 and more recent processor families do not provide these pins because the L2 cache is internal to the chip package.

14.5.2 Precedence of Cache Controls

The cache control flags and MTRRs operate hierarchically for restricting caching. That is, if the CD flag is set, caching is prevented globally (see Table 14-5). If the CD flag is clear, the page-level cache control flags and/or the MTRRs can be used to restrict caching. If there is an overlap of page-level and MTRR caching controls, the mechanism that prevents caching has precedence. For example, if an MTRR makes a region of system memory uncachable, a page-level caching control cannot be used to enable caching for a page in that region. The converse is also

true; that is, if a page-level caching control designates a page as uncacheable, an MTRR cannot be used to make the page cacheable.

In cases where there is an overlap in the assignment of the write-back and write-through caching policies to a page and a region of memory, the write-through policy takes precedence. The write-combining policy (which can only be assigned through an MTRR or the PAT) takes precedence over either write-through or write-back.

The selection of memory types at the page level varies depending on whether PAT is being used to select memory types for pages, as described in the following sections.

On processors based on Intel NetBurst microarchitecture, the third-level cache can be disabled by bit 6 of the IA32_MISC_ENABLE MSR. Using IA32_MISC_ENABLE[bit 6] takes precedence over the CD flag, MTRRs, and PAT for the L3 cache in those processors. That is, when the third-level cache disable flag is set (cache disabled), the other cache controls have no effect on the L3 cache; when the flag is clear (enabled), the cache controls have the same effect on the L3 cache as they have on the L1 and L2 caches.

IA32_MISC_ENABLE[bit 6] is not supported in Intel Core i7 processors, nor processors based on Intel Core, and Intel Atom microarchitectures.

14.5.2.1 Selecting Memory Types for Pentium Pro and Pentium II Processors

The Pentium Pro and Pentium II processors do not support the PAT. Here, the effective memory type for a page is selected with the MTRRs and the PCD and PWT bits in the page-table or page-directory entry for the page. Table 14-6 describes the mapping of MTRR memory types and page-level caching attributes to effective memory types, when normal caching is in effect (the CD and NW flags in control register CR0 are clear). Combinations that appear in gray are implementation-defined for the Pentium Pro and Pentium II processors. System designers are encouraged to avoid these implementation-defined combinations.

Table 14-6. Effective Page-Level Memory Type for Pentium Pro and Pentium II Processors

MTRR Memory Type ¹	PCD Value	PWT Value	Effective Memory Type
UC	X	X	UC
WC	0	0	WC
	0	1	WC
	1	0	WC
	1	1	UC
WT	0	X	WT
	1	X	UC
WP	0	0	WP
	0	1	WP
	1	0	WC
	1	1	UC
WB	0	0	WB
	0	1	WT
	1	X	UC

NOTE:

1. These effective memory types also apply to the Pentium 4, Intel Xeon, and Pentium III processors when the PAT bit is not used (set to 0) in page-table and page-directory entries.

When normal caching is in effect, the effective memory type shown in Table 14-6 is determined using the following rules:

1. If the PCD and PWT attributes for the page are both 0, then the effective memory type is identical to the MTRR-defined memory type.

2. If the PCD flag is set, then the effective memory type is UC.
3. If the PCD flag is clear and the PWT flag is set, the effective memory type is WT for the WB memory type and the MTRR-defined memory type for all other memory types.
4. Setting the PCD and PWT flags to opposite values is considered model-specific for the WP and WC memory types and architecturally-defined for the WB, WT, and UC memory types.

14.5.2.2 Selecting Memory Types for Pentium III and More Recent Processor Families

The Intel Core 2 Duo, Intel Atom, Intel Core Duo, Intel Core Solo, Pentium M, Pentium 4, Intel Xeon, and Pentium III processors use the PAT to select effective page-level memory types. Here, a memory type for a page is selected by the MTRRs and the value in a PAT entry that is selected with the PAT, PCD, and PWT bits in a page-table or page-directory entry (see Section 14.12.3, “Selecting a Memory Type from the PAT”). Table 14-7 describes the mapping of MTRR memory types and PAT entry types to effective memory types, when normal caching is in effect (the CD and NW flags in control register CR0 are clear).

Table 14-7. Effective Page-Level Memory Types for Pentium III and More Recent Processor Families

MTRR Memory Type	PAT Entry Value	Effective Memory Type
UC	UC	UC ¹
	UC-	UC ¹
	WC	WC
	WT	UC ¹
	WB	UC ¹
	WP	UC ¹
WC	UC	UC ²
	UC-	WC
	WC	WC
	WT	UC ^{2,3}
	WB	WC
	WP	UC ^{2,3}
WT	UC	UC ²
	UC-	UC ²
	WC	WC
	WT	WT
	WB	WT
	WP	WP ³
WB	UC	UC ²
	UC-	UC ²
	WC	WC
	WT	WT
	WB	WB
	WP	WP

Table 14-7. Effective Page-Level Memory Types for Pentium III and More Recent Processor Families (Contd.)

MTRR Memory Type	PAT Entry Value	Effective Memory Type
WP	UC	UC ²
	UC-	WC ³
	WC	WC
	WT	WT ³
	WB	WP
	WP	WP

NOTES:

1. The UC attribute comes from the MTRRs and the processors are not required to snoop their caches since the data could never have been cached. This attribute is preferred for performance reasons.
2. The UC attribute came from the page-table or page-directory entry and processors are required to check their caches because the data may be cached due to page aliasing, which is not recommended.
3. These combinations were specified as “undefined” in previous editions of the Intel® 64 and IA-32 Architectures Software Developer’s Manual. However, all processors that support both the PAT and the MTRRs determine the effective page-level memory types for these combinations as given.

14.5.2.3 Writing Values Across Pages with Different Memory Types

If two adjoining pages in memory have different memory types, and a word or longer operand is written to a memory location that crosses the page boundary between those two pages, the operand might be written to memory twice. This action does not present a problem for writes to actual memory; however, if a device is mapped to the memory space assigned to the pages, the device might malfunction.

14.5.3 Preventing Caching

To disable the L1, L2, and L3 caches after they have been enabled and have received cache fills, perform the following steps:

1. Enter the no-fill cache mode. (Set the CD flag in control register CR0 to 1 and the NW flag to 0.
2. Flush all caches using the WBINVD instruction.
3. Disable the MTRRs and set the default memory type to uncached or set all MTRRs for the uncached memory type (see the discussion of the discussion of the TYPE field and the E flag in Section 14.11.2.1, “IA32_MTRR_DEF_TYPE MSR”).

The caches must be flushed (step 2) after the CD flag is set to ensure system memory coherency. If the caches are not flushed, cache hits on reads will still occur and data will be read from valid cache lines.

The intent of the three separate steps listed above address three distinct requirements: (i) discontinue new data replacing existing data in the cache (ii) ensure data already in the cache are evicted to memory, (iii) ensure subsequent memory references observe UC memory type semantics. Different processor implementation of caching control hardware may allow some variation of software implementation of these three requirements. See note below.

NOTES

Setting the CD flag in control register CR0 modifies the processor’s caching behavior as indicated in Table 14-5, but setting the CD flag alone may not be sufficient across all processor families to force the effective memory type for all physical memory to be UC nor does it force strict memory ordering, due to hardware implementation variations across different processor families. To force the UC memory type and strict memory ordering on all of physical memory, it is sufficient to either program the MTRRs for all physical memory to be UC memory type or disable all MTRRs.

For the Pentium 4 and Intel Xeon processors, after the sequence of steps given above has been executed, the cache lines containing the code between the end of the WBINVD instruction and before the MTRRS have actually been disabled may be retained in the cache hierarchy. Here, to

remove code from the cache completely, a second WBINVD instruction must be executed after the MTRRs have been disabled.

For Intel Atom processors, setting the CD flag forces all physical memory to observe UC semantics (without requiring memory type of physical memory to be set explicitly). Consequently, software does not need to issue a second WBINVD as some other processor generations might require.

14.5.4 Disabling and Enabling the L3 Cache

On processors based on Intel NetBurst microarchitecture, the third-level cache can be disabled by bit 6 of the IA32_MISC_ENABLE MSR. The third-level cache disable flag (bit 6 of the IA32_MISC_ENABLE MSR) allows the L3 cache to be disabled and enabled, independently of the L1 and L2 caches. Prior to using this control to disable or enable the L3 cache, software should disable and flush all the processor caches, as described earlier in Section 14.5.3, “Preventing Caching,” to prevent loss of information stored in the L3 cache. After the L3 cache has been disabled or enabled, caching for the whole processor can be restored.

Newer Intel 64 processor with L3 do not support IA32_MISC_ENABLE[bit 6], the procedure described in Section 14.5.3, “Preventing Caching,” apply to the entire cache hierarchy.

14.5.5 Cache Management Instructions

The Intel 64 and IA-32 architectures provide several instructions for managing the L1, L2, and L3 caches. The INVD and WBINVD instructions are privileged instructions and operate on the L1, L2, and L3 caches as a whole. The PREFETCHh, CLFLUSH, and CLFLUSHOPT instructions and the non-temporal move instructions (MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, and MOVNTPD) offer more granular control over caching, and are available to all privileged levels.

The INVD and WBINVD instructions are used to invalidate the contents of the L1, L2, and L3 caches. The INVD instruction invalidates all internal cache entries, then generates a special-function bus cycle that indicates that external caches also should be invalidated. The INVD instruction should be used with care. It does not force a write-back of modified cache lines; therefore, data stored in the caches and not written back to system memory will be lost. Unless there is a specific requirement or benefit to invalidating the caches without writing back the modified lines (such as, during testing or fault recovery where cache coherency with main memory is not a concern), software should use the WBINVD instruction.

The WBINVD instruction first writes back any modified lines in all the internal caches, then invalidates the contents of the L1, L2, and L3 caches. It ensures that cache coherency with main memory is maintained regardless of the write policy in effect (that is, write-through or write-back). Following this operation, the WBINVD instruction generates one (P6 family processors) or two (Pentium and Intel486 processors) special-function bus cycles to indicate to external cache controllers that write-back of modified data followed by invalidation of external caches should occur. The amount of time or cycles for WBINVD to complete will vary due to the size of different cache hierarchies and other factors. As a consequence, the use of the WBINVD instruction can have an impact on interrupt/event response time.

The PREFETCHh instructions allow a program to suggest to the processor that a cache line from a specified location in system memory be prefetched into the cache hierarchy (see Section 14.8, “Explicit Caching”).

The CLFLUSH and CLFLUSHOPT instructions allow selected cache lines to be flushed from memory. These instructions give a program the ability to explicitly free up cache space, when it is known that cached section of system memory will not be accessed in the near future.

The non-temporal move instructions (MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, and MOVNTPD) allow data to be moved from the processor’s registers directly into system memory without being also written into the L1, L2, and/or L3 caches. These instructions can be used to prevent cache pollution when operating on data that is going to be modified only once before being stored back into system memory. These instructions operate on data in the general-purpose, MMX, and XMM registers.

14.5.6 L1 Data Cache Context Mode

L1 data cache context mode is a feature of processors based on the Intel NetBurst microarchitecture that support Intel Hyper-Threading Technology. When CPUID.01H:ECX[10] = 1, the processor supports setting L1 data cache context mode using the L1 data cache context mode flag (IA32_MISC_ENABLE[bit 24]). Selectable modes are adaptive mode (default) and shared mode.

The BIOS is responsible for configuring the L1 data cache context mode.

14.5.6.1 Adaptive Mode

Adaptive mode facilitates L1 data cache sharing between logical processors. When running in adaptive mode, the L1 data cache is shared across logical processors in the same core if:

- CR3 control registers for logical processors sharing the cache are identical.
- The same paging mode is used by logical processors sharing the cache.

In this situation, the entire L1 data cache is available to each logical processor (instead of being competitively shared).

If CR3 values are different for the logical processors sharing an L1 data cache or the logical processors use different paging modes, processors compete for cache resources. This reduces the effective size of the cache for each logical processor. Aliasing of the cache is not allowed (which prevents data thrashing).

14.5.6.2 Shared Mode

In shared mode, the L1 data cache is competitively shared between logical processors. This is true even if the logical processors use identical CR3 registers and paging modes.

In shared mode, linear addresses in the L1 data cache can be aliased, meaning that one linear address in the cache can point to different physical locations. The mechanism for resolving aliasing can lead to thrashing. For this reason, IA32_MISC_ENABLE[bit 24] = 0 is the preferred configuration for processors based on the Intel NetBurst microarchitecture that support Intel Hyper-Threading Technology.

14.6 SELF-MODIFYING CODE

A write to a memory location in a code segment that is currently cached in the processor causes the associated cache line (or lines) to be invalidated. This check is based on the physical address of the instruction. In addition, the P6 family and Pentium processors check whether a write to a code segment may modify an instruction that has been prefetched for execution. If the write affects a prefetched instruction, the prefetch queue is invalidated. This latter check is based on the linear address of the instruction. For the Pentium 4 and Intel Xeon processors, a write or a snoop of an instruction in a code segment, where the target instruction is already decoded and resident in the trace cache, invalidates the entire trace cache. The latter behavior means that programs that self-modify code can cause severe degradation of performance when run on the Pentium 4 and Intel Xeon processors.

In practice, the check on linear addresses should not create compatibility problems among IA-32 processors. Applications that include self-modifying code use the same linear address for modifying and fetching the instruction. Systems software, such as a debugger, that might possibly modify an instruction using a different linear address than that used to fetch the instruction, will execute a serializing operation, such as a CPUID instruction, before the modified instruction is executed, which will automatically resynchronize the instruction cache and prefetch queue. (See Section 11.1.3, "Handling Self- and Cross-Modifying Code," for more information about the use of self-modifying code.)

For Intel486 processors, a write to an instruction in the cache will modify it in both the cache and memory, but if the instruction was prefetched before the write, the old version of the instruction could be the one executed. To prevent the old instruction from being executed, flush the instruction prefetch unit by coding a jump instruction immediately after any write that modifies an instruction.

14.7 IMPLICIT CACHING (PENTIUM 4, INTEL® XEON®, AND P6 FAMILY PROCESSORS)

Implicit caching occurs when a memory element is made potentially cacheable, although the element may never have been accessed in the normal von Neumann sequence. Implicit caching occurs on the P6 and more recent processor families due to aggressive prefetching, branch prediction, and TLB miss handling. Implicit caching is an extension of the behavior of existing Intel386, Intel486, and Pentium processor systems, since software running on these processor families also has not been able to deterministically predict the behavior of instruction prefetch.

To avoid problems related to implicit caching, the operating system must explicitly invalidate the cache when changes are made to cacheable data that the cache coherency mechanism does not automatically handle. This includes writes to dual-ported or physically aliased memory boards that are not detected by the snooping mechanisms of the processor, and changes to page-table entries in memory.

The code in Example 14-1 shows the effect of implicit caching on page-table entries. The linear address F000H points to physical location B000H (the page-table entry for F000H contains the value B000H), and the page-table entry for linear address F000 is PTE_F000.

Example 14-1. Effect of Implicit Caching on Page-Table Entries

```
mov EAX, CR3; Invalidate the TLB
mov CR3, EAX; by copying CR3 to itself
mov PTE_F000, A000H; Change F000H to point to A000H
mov EBX, [F000H];
```

Because of speculative execution in the P6 and more recent processor families, the last MOV instruction performed would place the value at physical location B000H into EBX, rather than the value at the new physical address A000H. This situation is remedied by placing a TLB invalidation between the load and the store.

14.8 EXPLICIT CACHING

The Pentium III processor introduced four new instructions, the `PREFETCHh` instructions, that provide software with explicit control over the caching of data. These instructions provide “hints” to the processor that the data requested by a `PREFETCHh` instruction should be read into cache hierarchy now or as soon as possible, in anticipation of its use. The instructions provide different variations of the hint that allow selection of the cache level into which data will be read.

The `PREFETCHh` instructions can help reduce the long latency typically associated with reading data from memory and thus help prevent processor “stalls.” However, these instructions should be used judiciously. Overuse can lead to resource conflicts and hence reduce the performance of an application. Also, these instructions should only be used to prefetch data from memory; they should not be used to prefetch instructions. For more detailed information on the proper use of the prefetch instruction, refer to Chapter 7, “Optimizing Cache Usage,” in the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

14.9 INVALIDATING THE TRANSLATION LOOKASIDE BUFFERS (TLBS)

The processor updates its address translation caches (TLBs) transparently to software. Several mechanisms are available, however, that allow software and hardware to invalidate the TLBs either explicitly or as a side effect of another operation. Most details are given in Section 5.10.4, “Invalidation of TLBs and Paging-Structure Caches.” In addition, the following operations invalidate all TLB entries, irrespective of the setting of the G flag:

- Asserting or de-asserting the `FLUSH#` pin.
- (Pentium 4, Intel Xeon, and later processors only.) Writing to an MTRR (with a `WRMSR` instruction).
- Writing to control register `CR0` to modify the PG or PE flag.

- (Pentium 4, Intel Xeon, and later processors only.) Writing to control register CR4 to modify the PSE, PGE, or PAE flag.
- Writing to control register CR4 to change the PCIDE flag from 1 to 0.

See Section 5.10, “Caching Translation Information,” for additional information about the TLBs.

14.10 STORE BUFFER

Intel 64 and IA-32 processors temporarily store each write (store) to memory in a store buffer. The store buffer improves processor performance by allowing the processor to continue executing instructions without having to wait until a write to memory and/or to a cache is complete. It also allows writes to be delayed for more efficient use of memory-access bus cycles.

In general, the existence of the store buffer is transparent to software, even in systems that use multiple processors. The processor ensures that write operations are always carried out in program order. It also ensures that the contents of the store buffer are always drained to memory in the following situations:

- When an exception or interrupt is generated.
- (P6 and more recent processor families only) When a serializing instruction is executed.
- When an I/O instruction is executed.
- When a LOCK operation is performed.
- (P6 and more recent processor families only) When a BINIT operation is performed.
- (Pentium III, and more recent processor families only) When using an SFENCE instruction to order stores.
- (Pentium 4 and more recent processor families only) When using an MFENCE instruction to order stores.

The discussion of write ordering in Section 11.2, “Memory Ordering,” gives a detailed description of the operation of the store buffer.

14.11 MEMORY TYPE RANGE REGISTERS (MTRRS)

The following section pertains only to the P6 and more recent processor families.

The memory type range registers (MTRRs) provide a mechanism for associating the memory types (see Section 14.3, “Methods of Caching Available”) with physical-address ranges in system memory. They allow the processor to optimize operations for different types of memory such as RAM, ROM, frame-buffer memory, and memory-mapped I/O devices. They also simplify system hardware design by eliminating the memory control pins used for this function on earlier IA-32 processors and the external logic needed to drive them.

The MTRR mechanism allows multiple ranges to be defined in physical memory, and it defines a set of model-specific registers (MSRs) for specifying the type of memory that is contained in each range. Table 14-8 shows the memory types that can be specified and their properties; Figure 14-4 shows the mapping of physical memory with MTRRs. See Section 14.3, “Methods of Caching Available,” for a more detailed description of each memory type.

Following a hardware reset, the P6 and more recent processor families disable all the fixed and variable MTRRs, which in effect makes all of physical memory uncacheable. Initialization software should then set the MTRRs to a specific, system-defined memory map. Typically, the BIOS (basic input/output system) software configures the MTRRs. The operating system or executive is then free to modify the memory map using the normal page-level cacheability attributes.

In a multiprocessor system using a processor in the P6 family or a more recent family, each processor **MUST** use the identical MTRR memory map so that software will have a consistent view of memory.

NOTE

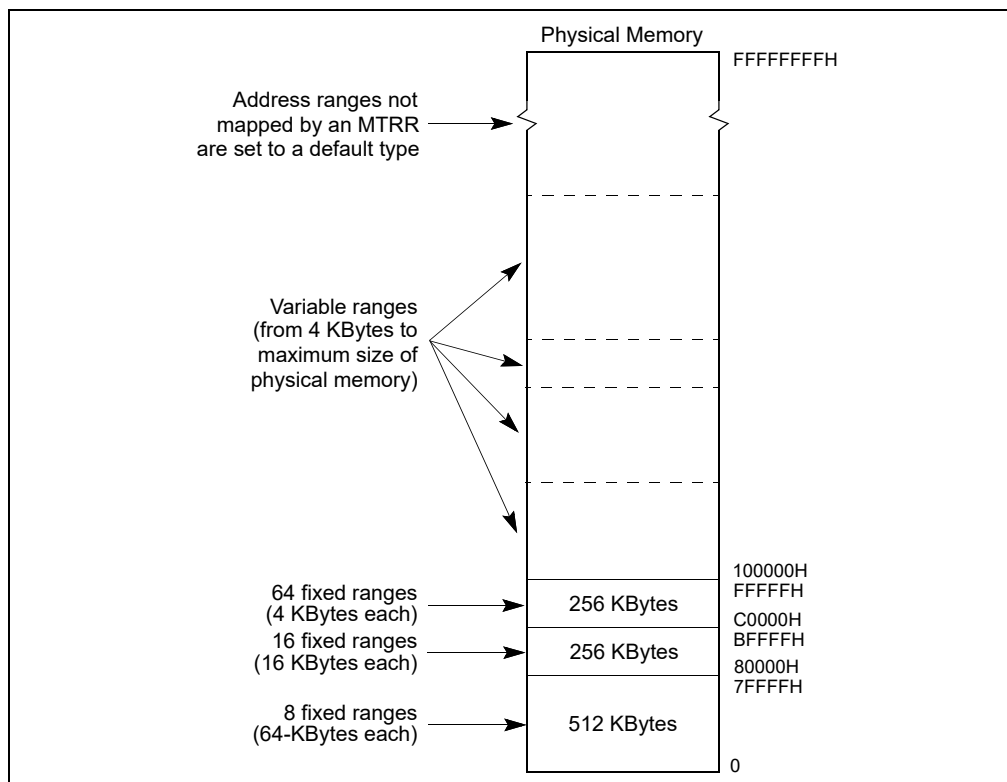
In multiple processor systems, the operating system must maintain MTRR consistency between all the processors in the system (that is, all processors must use the same MTRR values). The P6 and more recent processor families provide no hardware support for maintaining this consistency.

Table 14-8. Memory Types That Can Be Encoded in MTRRs

Memory Type and Mnemonic	Encoding in MTRR
Uncacheable (UC)	00H
Write Combining (WC)	01H
Reserved*	02H
Reserved*	03H
Write-through (WT)	04H
Write-protected (WP)	05H
Writeback (WB)	06H
Reserved*	7H through FFH

NOTE:

* Use of these encodings results in a general-protection exception (#GP).

**Figure 14-4. Mapping Physical Memory With MTRRs****14.11.1 MTRR Feature Identification**

The availability of the MTRR feature is model-specific. Software can determine if MTRRs are supported on a processor by executing the CPUID instruction and reading the state of the MTRR flag (bit 12) in the feature information register (EDX).

If the MTRR flag is set (indicating that the processor implements MTRRs), additional information about MTRRs can be obtained from the 64-bit IA32_MTRRCAP MSR (named MTRRcap MSR for the P6 family processors). The IA32_MTRRCAP MSR is a read-only MSR that can be read with the RDMSR instruction. Figure 14-5 shows the contents of the IA32_MTRRCAP MSR. The functions of the flags and field in this register are as follows:

- **VCNT (variable range registers count) field, bits 0 through 7** — Indicates the number of variable ranges implemented on the processor.
- **FIX (fixed range registers supported) flag, bit 8** — Fixed range MTRRs (IA32_MTRR_FIX64K_00000 through IA32_MTRR_FIX4K_0F8000) are supported when set; no fixed range registers are supported when clear.
- **WC (write combining) flag, bit 10** — The write-combining (WC) memory type is supported when set; the WC type is not supported when clear.
- **SMRR (System-Management Range Register) flag, bit 11** — The system-management range register (SMRR) interface is supported when bit 11 is set; the SMRR interface is not supported when clear.
- **PRMRR (Processor Reserved Memory Range Register) flag, bit 12** — The processor reserved memory range register (PRMRR) interface is supported when bit 12 is set; the PRMRR interface is not supported when clear.
- **SEAMRR (SEAM Range Register) flag, bit 15** — The SEAMRR interface to restrict access to a specified memory address range is supported when bit 15 is set.

Bit 9, bits 14:13, and bits 63:16 of the IA32_MTRRCAP MSR are not currently defined. If software attempts to write to the IA32_MTRRCAP MSR, a general-protection exception (#GP) is generated.

Software must read IA32_MTRRCAP VCNT field to determine the number of variable MTRRs and query other feature bits in IA32_MTRRCAP to determine additional capabilities that are supported in a processor. For example, some processors may report a value of '8' in the VCNT field, other processors may report VCNT with different values.

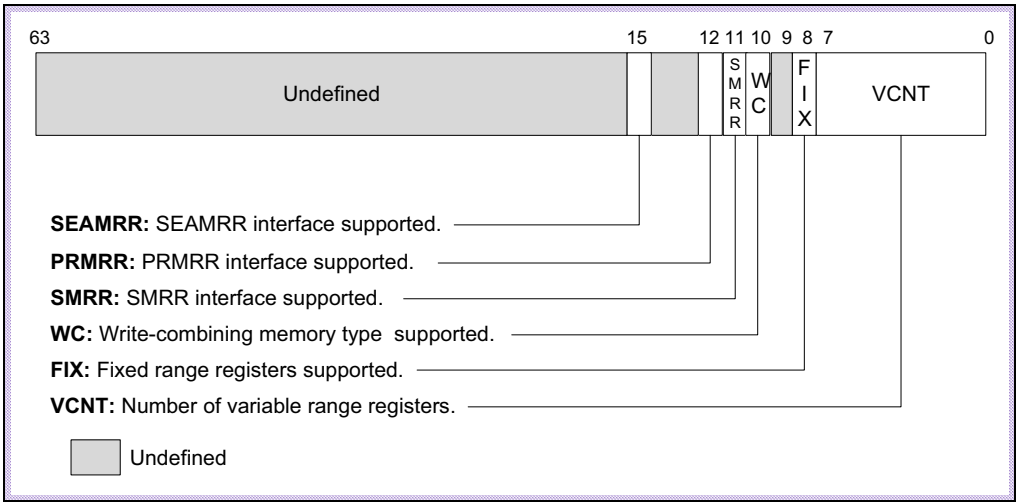


Figure 14-5. IA32_MTRRCAP Register

14.11.2 Setting Memory Ranges with MTRRs

The memory ranges and the types of memory specified in each range are set by three groups of registers: the IA32_MTRR_DEF_TYPE MSR, the fixed-range MTRRs, and the variable range MTRRs. These registers can be read and written to using the RDMSR and WRMSR instructions, respectively. The IA32_MTRRCAP MSR indicates the availability of these registers on the processor (see Section 14.11.1, “MTRR Feature Identification”).

14.11.2.1 IA32_MTRR_DEF_TYPE MSR

The IA32_MTRR_DEF_TYPE MSR (named MTRRdefType MSR for the P6 family processors) sets the default properties of the regions of physical memory that are not encompassed by MTRRs. The functions of the flags and field in this register are as follows:

- **Type field, bits 0 through 7** — Indicates the default memory type used for those physical memory address ranges that do not have a memory type specified for them by an MTRR (see Table 14-8 for the encoding of this

field). The legal values for this field are 0, 1, 4, 5, and 6. All other values result in a general-protection exception (#GP) being generated.

Intel recommends the use of the UC (uncached) memory type for all physical memory addresses where memory does not exist. To assign the UC type to nonexistent memory locations, it can either be specified as the default type in the Type field or be explicitly assigned with the fixed and variable MTRRs.

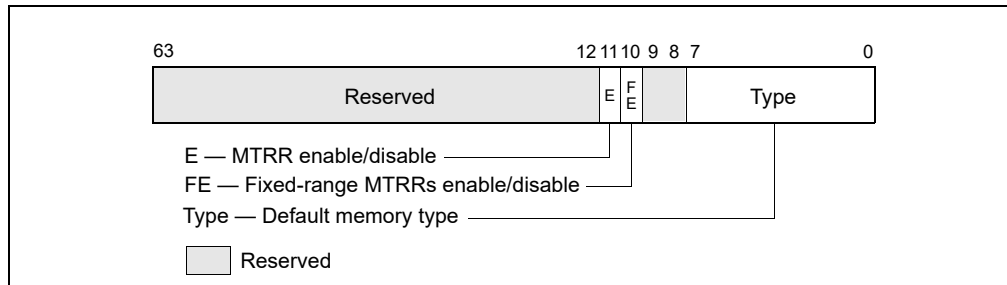


Figure 14-6. IA32_MTRR_DEF_TYPE MSR

- **FE (fixed MTRRs enabled) flag, bit 10** — Fixed-range MTRRs are enabled when set; fixed-range MTRRs are disabled when clear. When the fixed-range MTRRs are enabled, they take priority over the variable-range MTRRs when overlaps in ranges occur. If the fixed-range MTRRs are disabled, the variable-range MTRRs can still be used and can map the range ordinarily covered by the fixed-range MTRRs.
- **E (MTRRs enabled) flag, bit 11** — MTRRs are enabled when set; all MTRRs are disabled when clear, and the UC memory type is applied to all of physical memory. When this flag is set, the FE flag can disable the fixed-range MTRRs; when the flag is clear, the FE flag has no affect. When the E flag is set, the type specified in the default memory type field is used for areas of memory not already mapped by either a fixed or variable MTRR.

Bits 8 and 9, and bits 12 through 63, in the IA32_MTRR_DEF_TYPE MSR are reserved; the processor generates a general-protection exception (#GP) if software attempts to write nonzero values to them.

14.11.2.2 Fixed Range MTRRs

The fixed memory ranges are mapped with 11 fixed-range registers of 64 bits each. Each of these registers is divided into 8-bit fields that are used to specify the memory type for each of the sub-ranges the register controls:

- **Register IA32_MTRR_FIX64K_00000** — Maps the 512-KByte address range from 0H to 7FFFFH. This range is divided into eight 64-KByte sub-ranges.
- **Registers IA32_MTRR_FIX16K_80000 and IA32_MTRR_FIX16K_A0000** — Maps the two 128-KByte address ranges from 80000H to BFFFFH. This range is divided into sixteen 16-KByte sub-ranges, 8 ranges per register.
- **Registers IA32_MTRR_FIX4K_C0000 through IA32_MTRR_FIX4K_F8000** — Maps eight 32-KByte address ranges from C0000H to FFFFFH. This range is divided into sixty-four 4-KByte sub-ranges, 8 ranges per register.

Table 14-9 shows the relationship between the fixed physical-address ranges and the corresponding fields of the fixed-range MTRRs; Table 14-8 shows memory type encoding for MTRRs.

For the P6 family processors, the prefix for the fixed range MTRRs is MTRRfix.

14.11.2.3 Variable Range MTRRs

The Pentium 4, Intel Xeon, and P6 family processors permit software to specify the memory type for *m* variable-size address ranges, using a pair of MTRRs for each range. The number *m* of ranges supported is given in bits 7:0 of the IA32_MTRRCAP MSR (see Figure 14-5 in Section 14.11.1).

The first entry in each pair (IA32_MTRR_PHYSBASE_n) defines the base address and memory type for the range; the second entry (IA32_MTRR_PHYSMASK_n) contains a mask used to determine the address range. The “*n*” suffix is in the range 0 through *m*–1 and identifies a specific register pair.

For P6 family processors, the prefixes for these variable range MTRRs are MTRRphysBase and MTRRphysMask.

Table 14-9. Address Mapping for Fixed-Range MTRRs

Address Range (hexadecimal)								MTRR
63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
7000-7FFFF	6000-6FFFF	5000-5FFFF	4000-4FFFF	3000-3FFFF	2000-2FFFF	1000-1FFFF	0000-0FFFF	IA32_MTRR_FIX64K_00000
9C000-9FFFF	98000-9BFFF	94000-97FFF	90000-93FFF	8C000-8FFFF	88000-8BFFF	84000-87FFF	80000-83FFF	IA32_MTRR_FIX16K_80000
BC000-BFFFF	B8000-BBFFF	B4000-B7FFF	B0000-B3FFF	AC000-AFFFF	A8000-ABFFF	A4000-A7FFF	A0000-A3FFF	IA32_MTRR_FIX16K_A0000
C7000-C7FFF	C6000-C6FFF	C5000-C5FFF	C4000-C4FFF	C3000-C3FFF	C2000-C2FFF	C1000-C1FFF	C0000-C0FFF	IA32_MTRR_FIX4K_C0000
CF000-CFFFF	CE000-CEFFF	CD000-CDFFF	CC000-CCFFF	CB000-CBFFF	CA000-CAFFF	C9000-C9FFF	C8000-C8FFF	IA32_MTRR_FIX4K_C8000
D7000-D7FFF	D6000-D6FFF	D5000-D5FFF	D4000-D4FFF	D3000-D3FFF	D2000-D2FFF	D1000-D1FFF	D0000-D0FFF	IA32_MTRR_FIX4K_D0000
DF000-DFFFF	DE000-DEFFF	DD000-DDFFF	DC000-DCFFF	DB000-DBFFF	DA000-DAFFF	D9000-D9FFF	D8000-D8FFF	IA32_MTRR_FIX4K_D8000
E7000-E7FFF	E6000-E6FFF	E5000-E5FFF	E4000-E4FFF	E3000-E3FFF	E2000-E2FFF	E1000-E1FFF	E0000-E0FFF	IA32_MTRR_FIX4K_E0000
EF000-EFFFF	EE000-EEFFF	ED000-EDFFF	EC000-ECFFF	EB000-EBFFF	EA000-EAFFF	E9000-E9FFF	E8000-E8FFF	IA32_MTRR_FIX4K_E8000
F7000-F7FFF	F6000-F6FFF	F5000-F5FFF	F4000-F4FFF	F3000-F3FFF	F2000-F2FFF	F1000-F1FFF	F0000-F0FFF	IA32_MTRR_FIX4K_F0000
FF000-FFFFF	FE000-FEFFF	FD000-FDFFF	FC000-FCFFF	FB000-FBFFF	FA000-FAFFF	F9000-F9FFF	F8000-F8FFF	IA32_MTRR_FIX4K_F8000

Figure 14-7 shows flags and fields in these registers. The functions of these flags and fields are:

- **Type field, bits 0 through 7** — Specifies the memory type for the range (see Table 14-8 for the encoding of this field).
- **PhysBase field, bits MAXPHYADDR-1:12** — Specifies the base address of the address range.¹ This 24-bit value, in the case where MAXPHYADDR is 36 bits, is extended by 12 bits at the low end to form the base address (this automatically aligns the address on a 4-KByte boundary).
- **PhysMask field, bits MAXPHYADDR-1:12** — Specifies a mask (24 bits if the maximum physical address size is 36 bits, 28 bits if the maximum physical address size is 40 bits). The mask determines the range of the region being mapped, according to the following relationships:
 - $\text{Address\_Within\_Range AND PhysMask} = \text{PhysBase AND PhysMask}$
 - This value is extended by 12 bits at the low end to form the mask value. For more information: see Section 14.11.3, “Example Base and Mask Calculations.”
 - The width of the PhysMask field depends on the maximum physical address size supported by the processor. CPUID.80000008H reports the maximum physical address size supported by the processor. If CPUID.80000008H is not available, software may assume that the processor supports a 36-bit physical address size (then PhysMask is 24 bits wide and the upper 28 bits of IA32_MTRR_PHYSMASKn are reserved). See the Note below.

1. MAXPHYADDR is derived from the value enumerated by CPUID.80000008H:EAX[7:0]. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM. (For processors that do not support CPUID.80000008H, MAXPHYADDR is 36.)

- **V (valid) flag, bit 11** — Enables the register pair when set; disables register pair when clear.

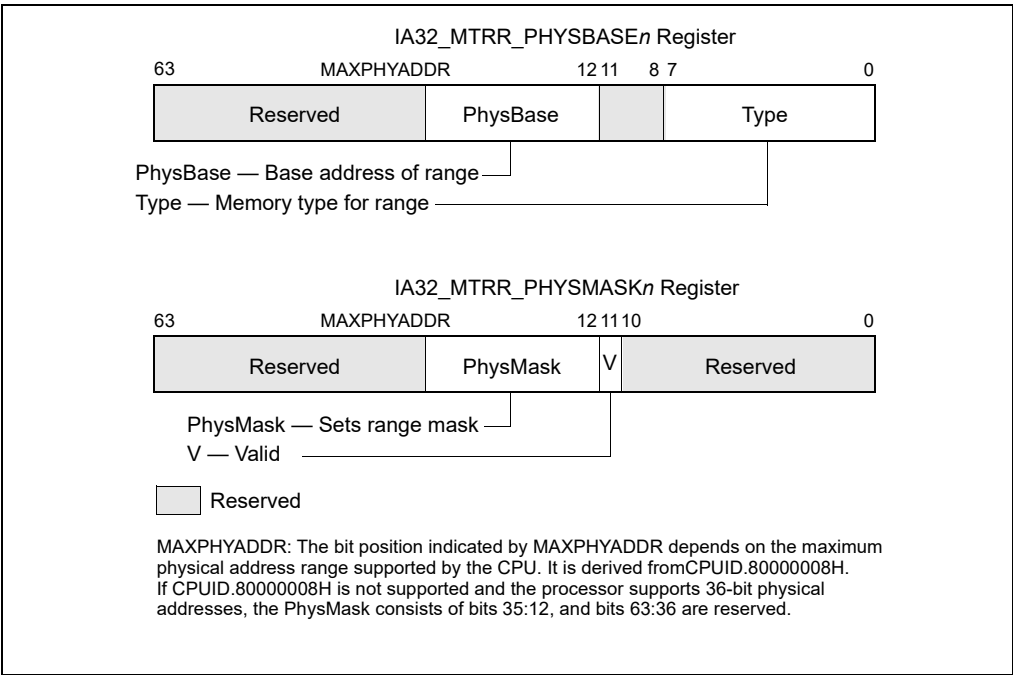


Figure 14-7. IA32_MTRR_PHYSBASE_n and IA32_MTRR_PHYSMASK_n Variable-Range Register Pair

All other bits in the IA32_MTRR_PHYSBASE_n and IA32_MTRR_PHYSMASK_n registers are reserved; the processor generates a general-protection exception (#GP) if software attempts to write to them.

Some mask values can result in ranges that are not continuous. In such ranges, the area not mapped by the mask value is set to the default memory type, unless some other MTRR specifies a type for that range. Intel does not encourage the use of “discontinuous” ranges.

NOTE

It is possible for software to parse the memory descriptions that BIOS provides by using the ACPI/INT15 e820 interface mechanism. This information then can be used to determine how MTRRs are initialized (for example: allowing the BIOS to define valid memory ranges and the maximum memory range supported by the platform, including the processor).

See Section 14.11.4.1, “MTRR Precedences,” for information on overlapping variable MTRR ranges.

14.11.2.4 System-Management Range Register Interface

If IA32_MTRRCAP[bit 11] is set, the processor supports the SMRR interface to restrict access to a specified memory address range used by system-management mode (SMM) software (see Section 34.4.2.1). If the SMRR interface is supported, SMM software is strongly encouraged to use it to protect the SMI code and data stored by SMI handler in the SMRAM region.

The system-management range registers consist of a pair of MSRs (see Figure 14-8). The IA32_SMRR_PHYSBASE MSR defines the base address for the SMRAM memory range and the memory type used to access it in SMM. The IA32_SMRR_PHYSMASK MSR contains a valid bit and a mask that determines the SMRAM address range protected by the SMRR interface. These MSRs may be written only in SMM; an attempt to write them outside of SMM causes a general-protection exception.¹

Figure 14-8 shows flags and fields in these registers. The functions of these flags and fields are the following:

1. For some processor models, these MSRs can be accessed by RDMSR and WRMSR only if the SMRR interface has been enabled using a model-specific bit in the IA32_FEATURE_CONTROL MSR.

- **Type field, bits 0 through 7** — Specifies the memory type for the range (see Table 14-8 for the encoding of this field).
- **PhysBase field, bits 12 through 31** — Specifies the base address of the address range. The address must be less than 4 GBytes and is automatically aligned on a 4-KByte boundary.
- **PhysMask field, bits 12 through 31** — Specifies a mask that determines the range of the region being mapped, according to the following relationships:
 - $\text{Address\_Within\_Range AND PhysMask} = \text{PhysBase AND PhysMask}$
 - This value is extended by 12 bits at the low end to form the mask value. For more information: see Section 14.11.3, “Example Base and Mask Calculations.”
- **V (valid) flag, bit 11** — Enables the register pair when set; disables register pair when clear.

Before attempting to access these SMRR registers, software must test bit 11 in the IA32_MTRRCAP register. If SMRR is not supported, reads from or writes to registers cause general-protection exceptions.

When the valid flag in the IA32_SMRR_PHYSMASK MSR is 1, accesses to the specified address range are treated as follows:

- If the logical processor is in SMM, accesses uses the memory type in the IA32_SMRR_PHYSBASE MSR.
- If the logical processor is not in SMM, write accesses are ignored and read accesses return a fixed value for each byte. The uncacheable memory type (UC) is used in this case.

The above items apply even if the address range specified overlaps with a range specified by the MTRRs.

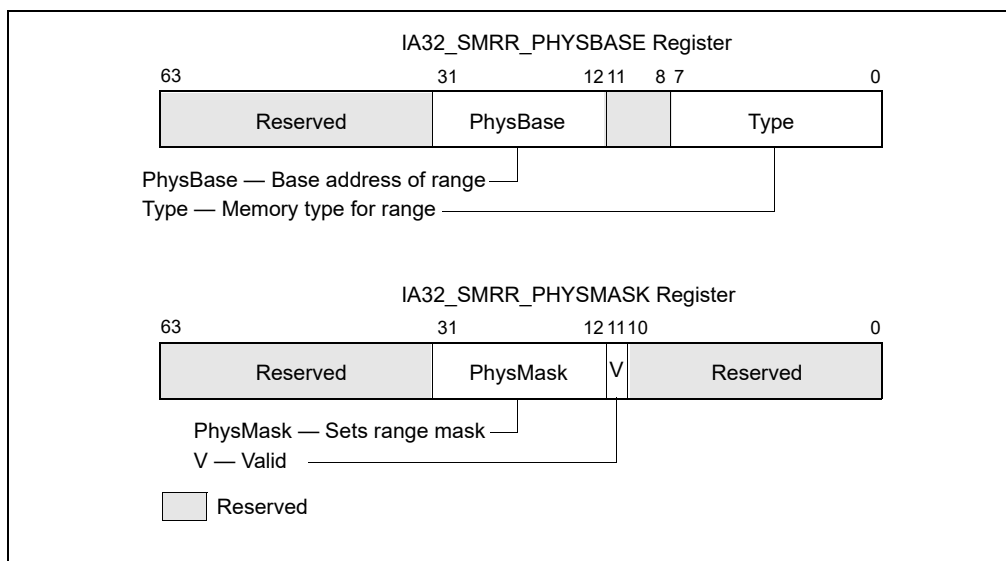


Figure 14-8. IA32_SMRR_PHYSBASE and IA32_SMRR_PHYSMASK SMRR Pair

14.11.3 Example Base and Mask Calculations

The examples in this section apply to processors that support a maximum physical address size of 36 bits. The base and mask values entered in variable-range MTRR pairs are 24-bit values that the processor extends to 36-bits.

For example, to enter a base address of 2 MBytes (200000H) in the IA32_MTRR_PHYSBASE3 register, the 12 least-significant bits are truncated and the value 000200H is entered in the PhysBase field. The same operation must be performed on mask values. For example, to map the address range from 200000H to 3FFFFFFH (2 MBytes to 4 MBytes), a mask value of FFFE0000H is required. Again, the 12 least-significant bits of this mask value are truncated, so that the value entered in the PhysMask field of IA32_MTRR_PHYSMASK3 is FFFE00H. This mask is chosen so that when any address in the 200000H to 3FFFFFFH range is ANDed with the mask value, it will return the same value as when the base address is ANDed with the mask value (which is 200000H).

To map the address range from 400000H to 7FFFFFFH (4 MBytes to 8 MBytes), a base value of 000400H is entered in the PhysBase field and a mask value of FFC00H is entered in the PhysMask field.

Example 14-2. Setting-Up Memory for a System

Here is an example of setting up the MTRRs for an system. Assume that the system has the following characteristics:

- 96 MBytes of system memory is mapped as write-back memory (WB) for highest system performance.
- A custom 4-MByte I/O card is mapped to uncached memory (UC) at a base address of 64 MBytes. This restriction forces the 96 MBytes of system memory to be addressed from 0 to 64 MBytes and from 68 MBytes to 100 MBytes, leaving a 4-MByte hole for the I/O card.
- An 8-MByte graphics card is mapped to write-combining memory (WC) beginning at address A0000000H.
- The BIOS area from 15 MBytes to 16 MBytes is mapped to UC memory.

The following settings for the MTRRs will yield the proper mapping of the physical address space for this system configuration.

```
IA32_MTRR_PHYSBASE0 = 0000 0000 0000 0006H
IA32_MTRR_PHYSMASK0 = 0000 000F FC00 0800H
Caches 0-64 MByte as WB cache type.
```

```
IA32_MTRR_PHYSBASE1 = 0000 0000 0400 0006H
IA32_MTRR_PHYSMASK1 = 0000 000F FE00 0800H
Caches 64-96 MByte as WB cache type.
```

```
IA32_MTRR_PHYSBASE2 = 0000 0000 0600 0006H
IA32_MTRR_PHYSMASK2 = 0000 000F FFC0 0800H
Caches 96-100 MByte as WB cache type.
```

```
IA32_MTRR_PHYSBASE3 = 0000 0000 0400 0000H
IA32_MTRR_PHYSMASK3 = 0000 000F FFC0 0800H
Caches 64-68 MByte as UC cache type.
```

```
IA32_MTRR_PHYSBASE4 = 0000 0000 00F0 0000H
IA32_MTRR_PHYSMASK4 = 0000 000F FFF0 0800H
Caches 15-16 MByte as UC cache type.
```

```
IA32_MTRR_PHYSBASE5 = 0000 0000 A000 0001H
IA32_MTRR_PHYSMASK5 = 0000 000F FF80 0800H
Caches A0000000-A0800000 as WC type.
```

This MTRR setup uses the ability to overlap any two memory ranges (as long as the ranges are mapped to WB and UC memory types) to minimize the number of MTRR registers that are required to configure the memory environment. This setup also fulfills the requirement that two register pairs are left for operating system usage.

14.11.3.1 Base and Mask Calculations for Greater-Than 36-bit Physical Address Support

For Intel 64 and IA-32 processors that support greater than 36 bits of physical address size, software should query CPUID.80000008H to determine the maximum physical address. See the footnote 1 on page 14-24.

Example 14-3. Setting-Up Memory for a System with a 40-Bit Address Size

If a processor supports 40-bits of physical address size, then the PhysMask field (in IA32_MTRR_PHYSMASK_n registers) is 28 bits instead of 24 bits. For this situation, Example 14-2 should be modified as follows:

```
IA32_MTRR_PHYSBASE0 = 0000 0000 0000 0006H
IA32_MTRR_PHYSMASK0 = 0000 00FF FC00 0800H
Caches 0-64 MByte as WB cache type.
```

IA32_MTRR_PHYSBASE1 = 0000 0000 0400 0006H

IA32_MTRR_PHYSMASK1 = 0000 00FF FE00 0800H

Caches 64-96 MByte as WB cache type.

IA32_MTRR_PHYSBASE2 = 0000 0000 0600 0006H

IA32_MTRR_PHYSMASK2 = 0000 00FF FFC0 0800H

Caches 96-100 MByte as WB cache type.

IA32_MTRR_PHYSBASE3 = 0000 0000 0400 0000H

IA32_MTRR_PHYSMASK3 = 0000 00FF FFC0 0800H

Caches 64-68 MByte as UC cache type.

IA32_MTRR_PHYSBASE4 = 0000 0000 00F0 0000H

IA32_MTRR_PHYSMASK4 = 0000 00FF FFF0 0800H

Caches 15-16 MByte as UC cache type.

IA32_MTRR_PHYSBASE5 = 0000 0000 A000 0001H

IA32_MTRR_PHYSMASK5 = 0000 00FF FF80 0800H

Caches A0000000-A0800000 as WC type.

14.11.4 Range Size and Alignment Requirement

A range that is to be mapped to a variable-range MTRR must meet the following “power of 2” size and alignment rules:

1. The minimum range size is 4 KBytes and the base address of the range must be on at least a 4-KByte boundary.
2. For ranges greater than 4 KBytes, each range must be of length 2^n and its base address must be aligned on a 2^n boundary, where n is a value equal to or greater than 12. The base-address alignment value cannot be less than its length. For example, an 8-KByte range cannot be aligned on a 4-KByte boundary. It must be aligned on at least an 8-KByte boundary.

14.11.4.1 MTRR Precedences

If the MTRRs are not enabled (by setting the E flag in the IA32_MTRR_DEF_TYPE MSR), then all memory accesses are of the UC memory type. If the MTRRs are enabled, then the memory type used for a memory access is determined as follows:

1. If the physical address falls within the first 1 MByte of physical memory and fixed MTRRs are enabled, the processor uses the memory type stored for the appropriate fixed-range MTRR.
2. Otherwise, the processor attempts to match the physical address with a memory type set by the variable-range MTRRs:
 - If one variable memory range matches, the processor uses the memory type stored in the IA32_MTRR_PHYSBASE n register for that range.
 - If two or more variable memory ranges match and the memory types are identical, then that memory type is used.
 - If two or more variable memory ranges match and one of the memory types is UC, the UC memory type is used.
 - If two or more variable memory ranges match and the memory types are WT and WB, the WT memory type is used.
 - For overlaps not defined by the above rules, processor behavior is undefined.
3. If no fixed or variable memory range matches, the processor uses the default memory type.

14.11.5 MTRR Initialization

On a hardware reset, the P6 and more recent processors clear the valid flags in variable-range MTRRs and clear the E flag in the IA32_MTRR_DEF_TYPE MSR to disable all MTRRs. All other bits in the MTRRs are undefined.

Prior to initializing the MTRRs, software (normally the system BIOS) must initialize all fixed-range and variable-range MTRR register fields to 0. Software can then initialize the MTRRs according to known types of memory, including memory on devices that it auto-configures. Initialization is expected to occur prior to booting the operating system.

See Section 14.11.8, “MTRR Considerations in MP Systems,” for information on initializing MTRRs in MP (multiple-processor) systems.

14.11.6 Remapping Memory Types

A system designer may re-map memory types to tune performance or because a future processor may not implement all memory types supported by the Pentium 4, Intel Xeon, and P6 family processors. The following rules support coherent memory-type re-mappings:

1. A memory type should not be mapped into another memory type that has a weaker memory ordering model. For example, the uncacheable type cannot be mapped into any other type, and the write-back, write-through, and write-protected types cannot be mapped into the weakly ordered write-combining type.
2. A memory type that does not delay writes should not be mapped into a memory type that does delay writes, because applications of such a memory type may rely on its write-through behavior. Accordingly, the write-back type cannot be mapped into the write-through type.
3. A memory type that views write data as not necessarily stored and read back by a subsequent read, such as the write-protected type, can only be mapped to another type with the same behavior (and there are no others for the Pentium 4, Intel Xeon, and P6 family processors) or to the uncacheable type.

In many specific cases, a system designer can have additional information about how a memory type is used, allowing additional mappings. For example, write-through memory with no associated write side effects can be mapped into write-back memory.

14.11.7 MTRR Maintenance Programming Interface

The operating system maintains the MTRRs after booting and sets up or changes the memory types for memory-mapped devices. The operating system should provide a driver and application programming interface (API) to access and set the MTRRs. The function calls `MemTypeGet()` and `MemTypeSet()` define this interface.

14.11.7.1 MemTypeGet() Function

The `MemTypeGet()` function returns the memory type of the physical memory range specified by the parameters base and size. The base address is the starting physical address and the size is the number of bytes for the memory range. The function automatically aligns the base address and size to 4-KByte boundaries. Pseudocode for the `MemTypeGet()` function is given in Example 14-4.

Example 14-4. MemTypeGet() Pseudocode

```

#define MIXED_TYPES -1    /* 0 < MIXED_TYPES || MIXED_TYPES > 256 */

IF CPU_FEATURES.MTRR /* processor supports MTRRs */
    THEN
        Align BASE and SIZE to 4-KByte boundary;
        IF (BASE + SIZE) wrap physical-address space
            THEN return INVALID;
        FI;
        IF MTRRdefType.E = 0
            THEN return UC;
        FI;
        FirstType := Get4KMemType (BASE);
        /* Obtains memory type for first 4-KByte range. */
        /* See Get4KMemType (4KByteRange) in Example 14-5. */
        FOR each additional 4-KByte range specified in SIZE
            NextType := Get4KMemType (4KByteRange);
            IF NextType != FirstType
                THEN return Mixed_Types;
            FI;
        ROF;
        return FirstType;
    ELSE return UNSUPPORTED;
FI;

```

If the processor does not support MTRRs, the function returns UNSUPPORTED. If the MTRRs are not enabled, then the UC memory type is returned. If more than one memory type corresponds to the specified range, a status of MIXED_TYPES is returned. Otherwise, the memory type defined for the range (UC, WC, WT, WB, or WP) is returned.

The pseudocode for the Get4KMemType() function in Example 14-5 obtains the memory type for a single 4-KByte range at a given physical address. The sample code determines whether an PHY_ADDRESS falls within a fixed range by comparing the address with the known fixed ranges: 0 to 7FFFFH (64-KByte regions), 80000H to BFFFFH (16-KByte regions), and C0000H to FFFFFH (4-KByte regions). If an address falls within one of these ranges, the appropriate bits within one of its MTRRs determine the memory type.

Example 14-5. Get4KMemType() Pseudocode

```

IF IA32_MTRRCAP.FIX AND MTRRdefType.FE /* fixed registers enabled */
    THEN IF PHY_ADDRESS is within a fixed range
        return IA32_MTRR_FIX.Type;
FI;
FOR each variable-range MTRR in IA32_MTRRCAP.VCNT
    IF IA32_MTRR_PHYSMASK.V = 0
        THEN continue;
    FI;
    IF (PHY_ADDRESS AND IA32_MTRR_PHYSMASK.Mask) =
        (IA32_MTRR_PHYSBASE.Base
         AND IA32_MTRR_PHYSMASK.Mask)
        THEN
            return IA32_MTRR_PHYSBASE.Type;
    FI;
ROF;
return MTRRdefType.Type;

```

14.11.7.2 MemTypeSet() Function

The MemTypeSet() function in Example 14-6 sets a MTRR for the physical memory range specified by the parameters base and size to the type specified by type. The base address and size are multiples of 4 KBytes and the size is not 0.

Example 14-6. MemTypeSet Pseudocode

```

IF CPU_FEATURES.MTRR (* processor supports MTRRs *)
  THEN
    IF BASE and SIZE are not 4-KByte aligned or size is 0
      THEN return INVALID;
    FI;
    IF (BASE + SIZE) wrap 4-GByte address space
      THEN return INVALID;
    FI;
    IF TYPE is invalid for Pentium 4, Intel Xeon, and P6 family
    processors
      THEN return UNSUPPORTED;
    FI;
    IF TYPE is WC and not supported
      THEN return UNSUPPORTED;
    FI;
    IF IA32_MTRRCAP.FIX is set AND range can be mapped using a
    fixed-range MTRR
      THEN
        pre_mtrr_change();
        update affected MTRR;
        post_mtrr_change();
      FI;

  ELSE (* try to map using a variable MTRR pair *)
    IF IA32_MTRRCAP.VCNT = 0
      THEN return UNSUPPORTED;
    FI;
    IF conflicts with current variable ranges
      THEN return RANGE_OVERLAP;
    FI;
    IF no MTRRs available
      THEN return VAR_NOT_AVAILABLE;
    FI;
    IF BASE and SIZE do not meet the power of 2 requirements for
    variable MTRRs
      THEN return INVALID_VAR_REQUEST;
    FI;
    pre_mtrr_change();
    Update affected MTRRs;
    post_mtrr_change();
  FI;

pre_mtrr_change()
  BEGIN
    disable interrupts;
    Save current value of CR4;
    disable and flush caches;
  
```

```

        flush TLBs;
        disable MTRRs;
        IF multiprocessing
            THEN maintain consistency through IPIs;
        FI;
    END
post_mtrr_change()
BEGIN
    flush caches and TLBs;
    enable MTRRs;
    enable caches;
    restore value of CR4;
    enable interrupts;
END

```

The physical address to variable range mapping algorithm in the MemTypeSet function detects conflicts with current variable range registers by cycling through them and determining whether the physical address in question matches any of the current ranges. During this scan, the algorithm can detect whether any current variable ranges overlap and can be concatenated into a single range.

The pre_mtrr_change() function disables interrupts prior to changing the MTRRs, to avoid executing code with a partially valid MTRR setup. The algorithm disables caching by setting the CD flag and clearing the NW flag in control register CR0. The caches are invalidated using the WBINVD instruction. The algorithm flushes all TLB entries either by clearing the page-global enable (PGE) flag in control register CR4 (if PGE was already set) or by updating control register CR3 (if PGE was already clear). Finally, it disables MTRRs by clearing the E flag in the IA32_MTRR_DEF_TYPE MSR.

After the memory type is updated, the post_mtrr_change() function re-enables the MTRRs and again invalidates the caches and TLBs. This second invalidation is required because of the processor's aggressive prefetch of both instructions and data. The algorithm restores interrupts and re-enables caching by setting the CD flag.

An operating system can batch multiple MTRR updates so that only a single pair of cache invalidations occur.

14.11.8 MTRR Considerations in MP Systems

In MP (multiple-processor) systems, the operating systems must maintain MTRR consistency between all the processors in the system. The Pentium 4, Intel Xeon, and P6 family processors provide no hardware support to maintain this consistency. In general, all processors must have the same MTRR values.

This requirement implies that when the operating system initializes an MP system, it must load the MTRRs of the boot processor while the E flag in register MTRRdefType is 0. The operating system then directs other processors to load their MTRRs with the same memory map. After all the processors have loaded their MTRRs, the operating system signals them to enable their MTRRs. Barrier synchronization is used to prevent further memory accesses until all processors indicate that the MTRRs are enabled. This synchronization is likely to be a shoot-down style algorithm, with shared variables and interprocessor interrupts.

Any change to the value of the MTRRs in an MP system requires the operating system to repeat the loading and enabling process to maintain consistency, using the following procedure:

1. Broadcast to all processors to execute the following code sequence.
2. Disable interrupts.
3. Wait for all processors to reach this point.
4. Enter the no-fill cache mode. (Set the CD flag in control register CR0 to 1 and the NW flag to 0.)
5. Flush all caches using the WBINVD instructions. On a processor that supports self-snooping (enumerating CPUID.01H:EDX.SELF_SNOOP[27] as 1), this step may be unnecessary. However, if there are changes for which self-snooping behavior would be problematic (e.g., changing the memory type of a cache line from WB to UC for memory-mapped I/O), execution of WBINVD would still be required.

6. If either the PGE or PCIDE flag is set in control register CR4, flush all TLBs by clearing one or both of these flags.
7. If the PGE and PCIDE flags are both clear in control register CR4, flush all TLBs by executing a MOV from control register CR3 to another register and then a MOV from that register back to CR3.
8. Disable all range registers (by clearing the E flag in register MTRRdefType). If only variable ranges are being modified, software may clear the valid bits for the affected register pairs instead.
9. Update the MTRRs.
10. Enable all range registers (by setting the E flag in register MTRRdefType). If only variable-range registers were modified and their individual valid bits were cleared, then set the valid bits for the affected ranges instead.
11. Flush all caches and all TLBs a second time. (The TLB flush is required for Pentium 4, Intel Xeon, and P6 family processors. Executing the WBINVD instruction is not needed when using Pentium 4, Intel Xeon, and P6 family processors, but it may be needed in future systems.)
12. Enter the normal cache mode to re-enable caching. (Set the CD and NW flags in control register CR0 to 0.)
13. Restore the values of the PGE and/or PCIDE flags in control register CR4, if cleared in Step 6 (above).
14. Wait for all processors to reach this point.
15. Enable interrupts.

14.11.9 Large Page Size Considerations

The MTRRs provide memory typing for a limited number of regions that have a 4 KByte granularity (the same granularity as 4-KByte pages). The memory type for a given page is cached in the processor's TLBs. When using large pages (2 MBytes, 4 MBytes, or 1 GBytes), a single page-table entry covers multiple 4-KByte granules, each with a single memory type. Because the memory type for a large page is cached in the TLB, the processor can behave in an undefined manner if a large page is mapped to a region of memory that MTRRs have mapped with multiple memory types.

Undefined behavior can be avoided by ensuring that all MTRR memory-type ranges within a large page are of the same type. If a large page maps to a region of memory containing different MTRR-defined memory types, the PCD and PWT flags in the page-table entry should be set for the most conservative memory type for that range. For example, a large page used for memory mapped I/O and regular memory is mapped as UC memory. Alternatively, the operating system can map the region using multiple 4-KByte pages each with its own memory type.

The requirement that all 4-KByte ranges in a large page are of the same memory type implies that large pages with different memory types may suffer a performance penalty, since they must be marked with the lowest common denominator memory type. The same consideration apply to 1 GByte pages, each of which may consist of multiple 2-Mbyte ranges.

The Pentium 4, Intel Xeon, and P6 family processors provide special support for the physical memory range from 0 to 4 MBytes, which is potentially mapped by both the fixed and variable MTRRs. This support is invoked when a Pentium 4, Intel Xeon, or P6 family processor detects a large page overlapping the first 1 MByte of this memory range with a memory type that conflicts with the fixed MTRRs. Here, the processor maps the memory range as multiple 4-KByte pages within the TLB. This operation ensures correct behavior at the cost of performance. To avoid this performance penalty, operating-system software should reserve the large page option for regions of memory at addresses greater than or equal to 4 MBytes.

14.12 PAGE ATTRIBUTE TABLE (PAT)

The Page Attribute Table (PAT) extends the IA-32 architecture's page-table format to allow memory types to be assigned to regions of physical memory based on linear address mappings. The PAT is a companion feature to the MTRRs; that is, the MTRRs allow mapping of memory types to regions of the physical address space, where the PAT allows mapping of memory types to pages within the linear address space. The MTRRs are useful for statically describing memory types for physical ranges, and are typically set up by the system BIOS. The PAT extends the functions of the PCD and PWT bits in page tables to allow all five of the memory types that can be assigned with the MTRRs (plus one additional memory type) to also be assigned dynamically to pages of the linear address space.

The PAT was introduced to IA-32 architecture on the Pentium III processor. It is also available in the Pentium 4 and Intel Xeon processors.

14.12.1 Detecting Support for the PAT Feature

An operating system or executive can detect the availability of the PAT by executing the CPUID instruction with a value of 1 in the EAX register. Support for the PAT is indicated by the PAT flag (bit 16 of the values returned to EDX register). If the PAT is supported, the operating system or executive can use the IA32_PAT MSR to program the PAT. When memory types have been assigned to entries in the PAT, software can then use of the PAT-index bit (PAT) in the page-table and page-directory entries along with the PCD and PWT bits to assign memory types from the PAT to individual pages.

Note that there is no separate flag or control bit in any of the control registers that enables the PAT. The PAT is always enabled on all processors that support it, and the table lookup always occurs whenever paging is enabled, in all paging modes.

14.12.2 IA32_PAT MSR

The IA32_PAT MSR is located at MSR address 277H (see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4). Figure 14-9. shows the format of the 64-bit IA32_PAT MSR.

The IA32_PAT MSR contains eight page attribute fields: PA0 through PA7. The three low-order bits of each field are used to specify a memory type. The five high-order bits of each field are reserved, and must be set to all 0s. Each of the eight page attribute fields can contain any of the memory type encodings specified in Table 14-10.

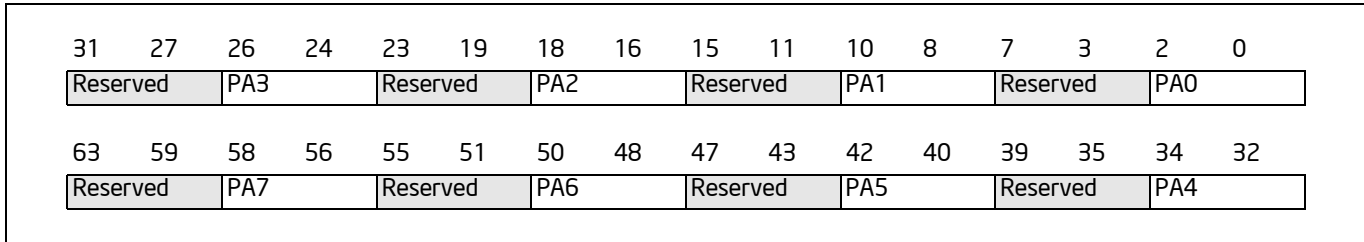


Figure 14-9. IA32_PAT MSR

Note that for the P6 family processors, the IA32_PAT MSR is named the PAT MSR.

Table 14-10. Memory Types That Can Be Encoded With PAT

Encoding	Mnemonic
00H	Uncacheable (UC)
01H	Write Combining (WC)
02H	Reserved*
03H	Reserved*
04H	Write Through (WT)
05H	Write Protected (WP)
06H	Write Back (WB)
07H	Uncached (UC-)
08H - FFH	Reserved*

NOTE:
* Using these encodings will result in a general-protection exception (#GP).

14.12.3 Selecting a Memory Type from the PAT

To select a memory type for a page from the PAT, a 3-bit index made up of the PAT, PCD, and PWT bits must be encoded in the page-table or page-directory entry for the page. Table 14-11 shows the possible encodings of the PAT, PCD, and PWT bits and the PAT entry selected with each encoding. The PAT bit is bit 7 in page-table entries that point to 4-KByte pages and bit 12 in paging-structure entries that point to larger pages. The PCD and PWT bits are bits 4 and 3, respectively, in paging-structure entries that point to pages of any size.

The PAT entry selected for a page is used in conjunction with the MTRR setting for the region of physical memory in which the page is mapped to determine the effective memory type for the page, as shown in Table 14-7.

Table 14-11. Selection of PAT Entries with PAT, PCD, and PWT Flags

PAT	PCD	PWT	PAT Entry
0	0	0	PAT0
0	0	1	PAT1
0	1	0	PAT2
0	1	1	PAT3
1	0	0	PAT4
1	0	1	PAT5
1	1	0	PAT6
1	1	1	PAT7

14.12.4 Programming the PAT

Table 14-12 shows the default setting for each PAT entry following a power up or reset of the processor. The setting remain unchanged following a soft reset (INIT reset).

Table 14-12. Memory Type Setting of PAT Entries Following a Power-up or Reset

PAT Entry	Memory Type Following Power-up or Reset
PAT0	WB
PAT1	WT
PAT2	UC-
PAT3	UC
PAT4	WB
PAT5	WT
PAT6	UC-
PAT7	UC

The values in all the entries of the PAT can be changed by writing to the IA32_PAT MSR using the WRMSR instruction. The IA32_PAT MSR is read and write accessible (use of the RDMSR and WRMSR instructions, respectively) to software operating at a CPL of 0. Table 14-10 shows the allowable encoding of the entries in the PAT. Attempting to write an undefined memory type encoding into the PAT causes a general-protection (#GP) exception to be generated.

The operating system (OS) is responsible for ensuring that changes to a PAT entry occur in a manner that maintains the consistency of the processor caches and translation lookaside buffers (TLB). It requires the OS to invalidate all affected TLB entries (including global entries) and all entries in all paging-structure caches. It may also require flushing of the processor caches in certain situations. This can be accomplished in various ways, including the sequence below or by following the procedure specified in Section 14.11.8, "MTRR Considerations in MP Systems." (See Section 5.10.4, "Invalidation of TLBs and Paging-Structure Caches" for additional background information.) Also note that in a multi-processor environment, it is the software's responsibility to resolve differences in conflicting memory types across logical processors that may arise from changes to the PAT (e.g., if two

logical processors map a linear address to the same physical address but have PATs that specify a different memory type for that physical address).

Example of a sequence to invalidate the processor TLBs and caches (if necessary):

1. If the PCIDE or PGE flag is set in CR4, flush TLBs by clearing one of those flags (then restore the flag via a subsequent CR4 write).
Otherwise, flush TLBs by executing a MOV from control register CR3 to another register and then a MOV from that register back to CR3.
2. In the case that there are changes to memory-type mappings for which cache self-snooping behavior would be problematic given the existing mappings (e.g., changing a cache line's memory type from WB to UC to be used for memory-mapped I/O), then cache flushing is also required. This can be done by executing CLFLUSH operations for all affected cache lines or by executing the WBINVD instruction (recommended only if there are a large number of affected mappings or if it is unknown which mappings are affected).

The PAT allows any memory type to be specified in the page tables, and therefore it is possible to have a single physical page mapped to two or more different linear addresses, each with different memory types. Intel does not support this practice because it may lead to undefined operations that can result in a system failure. In particular, a WC page must never be aliased to a cacheable page because WC writes may not check the processor caches.

When remapping a page that was previously mapped as a cacheable memory type to a WC page, an operating system can avoid this type of aliasing by doing the following:

1. Remove the previous mapping to a cacheable memory type in the page tables; that is, make them not present.
2. Flush the TLBs of processors that may have used the mapping, even speculatively.
3. Create a new mapping to the same physical address with a new memory type, for instance, WC.
4. Flush the caches on all processors that may have used the mapping previously. Note on processors that support self-snooping, CPUID feature flag bit 27, this step is unnecessary.

Operating systems that use a page directory as a page table (to map large pages) and enable page size extensions must carefully scrutinize the use of the PAT index bit for the 4-KByte page-table entries. The PAT index bit for a page-table entry (bit 7) corresponds to the page size bit in a page-directory entry. Therefore, the operating system can only use PAT entries PA0 through PA3 when setting the caching type for a page table that is also used as a page directory. If the operating system attempts to use PAT entries PA4 through PA7 when using this memory as a page table, it effectively sets the PS bit for the access to this memory as a page directory.

For compatibility with earlier IA-32 processors that do not support the PAT, care should be taken in selecting the encodings for entries in the PAT (see Section 14.12.5, "PAT Compatibility with Earlier IA-32 Processors").

14.12.5 PAT Compatibility with Earlier IA-32 Processors

For IA-32 processors that support the PAT, the IA32_PAT MSR is always active. That is, the PCD and PWT bits in page-table entries and in page-directory entries (that point to pages) are always select a memory type for a page indirectly by selecting an entry in the PAT. They never select the memory type for a page directly as they do in earlier IA-32 processors that do not implement the PAT (see Table 14-6).

To allow compatibility for code written to run on earlier IA-32 processor that do not support the PAT, the PAT mechanism has been designed to allow backward compatibility to earlier processors. This compatibility is provided through the ordering of the PAT, PCD, and PWT bits in the 3-bit PAT entry index. For processors that do not implement the PAT, the PAT index bit (bit 7 in the page-table entries and bit 12 in the page-directory entries) is reserved and set to 0. With the PAT bit reserved, only the first four entries of the PAT can be selected with the PCD and PWT bits. At power-up or reset (see Table 14-12), these first four entries are encoded to select the same memory types as the PCD and PWT bits would normally select directly in an IA-32 processor that does not implement the PAT. So, if encodings of the first four entries in the PAT are left unchanged following a power-up or reset, code written to run on earlier IA-32 processors that do not implement the PAT will run correctly on IA-32 processors that do implement the PAT.

19. Updates to Chapter 20, Volume 3B

Change bars and text show changes to Chapter 20 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- In Figure 20-21, "L3 Cache Monitoring Capability Enumeration Data (CPUID.0FH.01H)" (Section 20.18.5), the Upscaling Factor was corrected to register EBX.
- Added new Section 20.20.3, "Asymmetrical Enumeration of Intel® RDT Features".

CHAPTER 20

DEBUG, BRANCH PROFILE, TSC, AND INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) FEATURES

NOTE

This chapter makes numerous references to last-branch recording (LBR) facilities. Unless noted otherwise, all such references in this chapter are to an earlier non-architectural form of the feature. Chapter 21 defines an architectural form of last-branch recording that is supported on newer processors.

Intel 64 and IA-32 architectures provide debug facilities for use in debugging code and monitoring performance. These facilities are valuable for debugging application software, system software, and multitasking operating systems. Debug support is accessed using debug registers (DR0 through DR7) and model-specific registers (MSRs):

- Debug registers hold the addresses of memory and I/O locations called breakpoints. Breakpoints are user-selected locations in a program, a data-storage area in memory, or specific I/O ports. They are set where a programmer or system designer wishes to halt execution of a program and examine the state of the processor by invoking debugger software. A debug exception (#DB) is generated when a memory or I/O access is made to a breakpoint address.
- MSRs monitor branches, interrupts, and exceptions; they record addresses of the last branch, interrupt or exception taken and the last branch taken before an interrupt or exception.
- Time stamp counter is described in Section 20.17, "Time-Stamp Counter."
- Features that allow monitoring of shared platform resources such as the L3 cache are described in Section 20.18, "Intel® Resource Director Technology (Intel® RDT) Monitoring Features."
- Features that enable control over shared platform resources are described in Section 20.19, "Intel® Resource Director Technology (Intel® RDT) Allocation Features."
- Features that enable control over shared platform resources for non-CPU agents are described in Section 20.20, "Intel® Resource Director Technology (Intel® RDT) for Non-CPU Agents."¹

20.1 OVERVIEW OF DEBUG SUPPORT FACILITIES

The following processor facilities support debugging and performance monitoring:

- **Debug exception (#DB)** — Transfers program control to a debug procedure or task when a debug event occurs.
- **Breakpoint exception (#BP)** — See breakpoint instruction (INT3) below.
- **Breakpoint-address registers (DR0 through DR3)** — Specifies the addresses of up to 4 breakpoints.
- **Debug status register (DR6)** — Reports the conditions that were in effect when a debug or breakpoint exception was generated.
- **Debug control register (DR7)** — Specifies the forms of memory or I/O access that cause breakpoints to be generated.
- **T (trap) flag, TSS** — Generates a debug exception (#DB) when an attempt is made to switch to a task with the T flag set in its TSS.
- **RF (resume) flag, EFLAGS register** — Suppresses multiple exceptions to the same instruction.
- **TF (trap) flag, EFLAGS register** — Generates a debug exception (#DB) after every execution of an instruction.

1. Additional information about Intel® RDT can be found in the document titled "Intel® Resource Director Technology Architecture Specification," available here: <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>.

- **Breakpoint instruction (INT3)** — Generates a breakpoint exception (#BP) that transfers program control to the debugger procedure or task. This instruction is an alternative way to set instruction breakpoints. It is especially useful when more than four breakpoints are desired, or when breakpoints are being placed in the source code.
- **Last branch recording facilities** — Store branch records in the last branch record (LBR) stack MSRs for the most recent taken branches, interrupts, and/or exceptions in MSRs. A branch record consist of a branch-from and a branch-to instruction address. Send branch records out on the system bus as branch trace messages (BTMs).

These facilities allow a debugger to be called as a separate task or as a procedure in the context of the current program or task. The following conditions can be used to invoke the debugger:

- Task switch to a specific task.
- Execution of the breakpoint instruction.
- Execution of any instruction.
- Execution of an instruction at a specified address.
- Read or write to a specified memory address/range.
- Write to a specified memory address/range.
- Input from a specified I/O address/range.
- Output to a specified I/O address/range.
- Attempt to change the contents of a debug register.

20.2 DEBUG REGISTERS

Eight debug registers (see Figure 20-1 for 32-bit operation and Figure 20-2 for 64-bit operation) control the debug operation of the processor. These registers can be written to and read using the move to/from debug register form of the MOV instruction. A debug register may be the source or destination operand for one of these instructions.

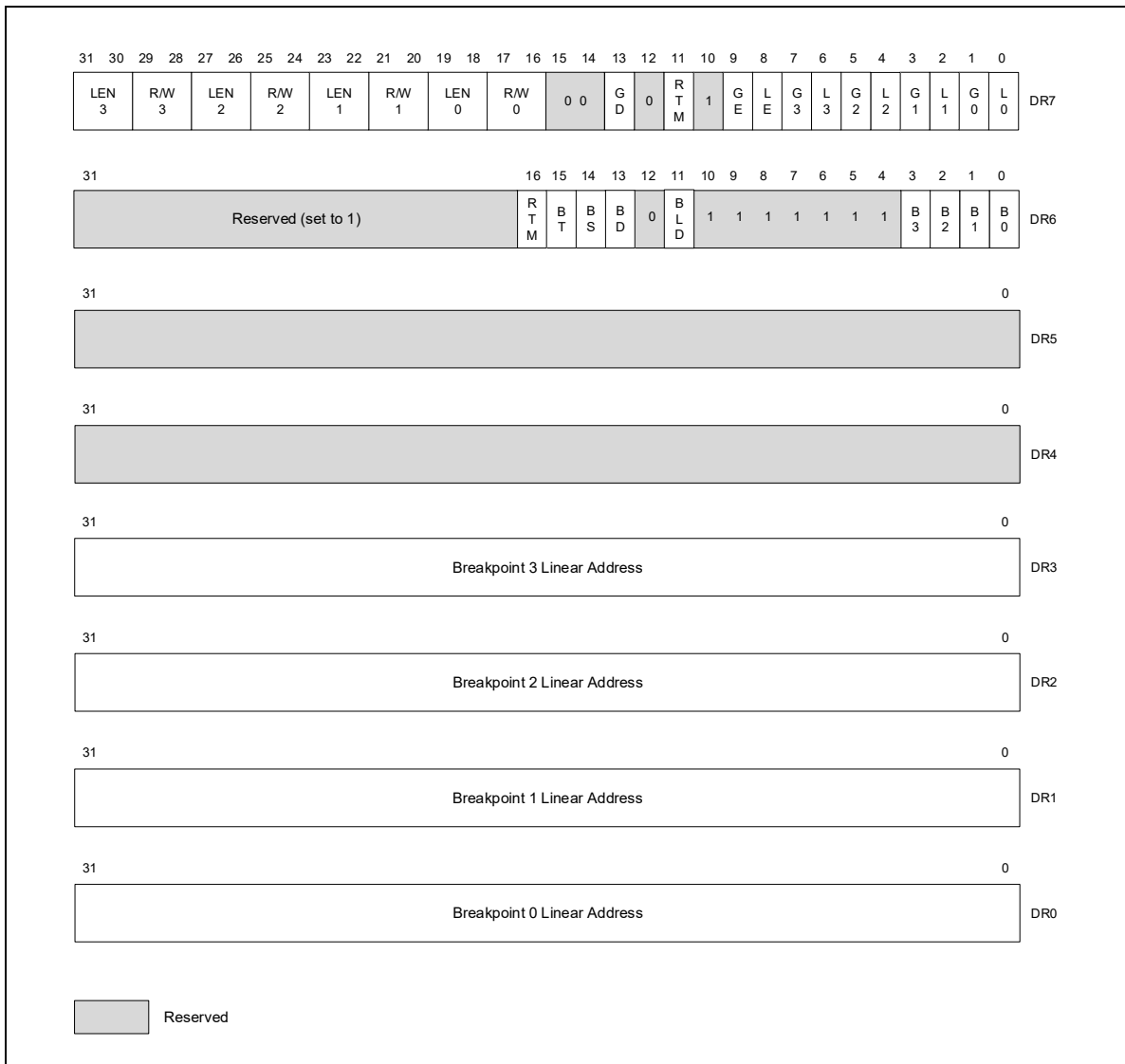


Figure 20-1. Debug Registers

Debug registers are privileged resources; a MOV instruction that accesses these registers can only be executed in real-address mode, in SMM or in protected mode at a CPL of 0. An attempt to read or write the debug registers from any other privilege level generates a general-protection exception (#GP).

The primary function of the debug registers is to set up and monitor from 1 to 4 breakpoints, numbered 0 through 3. For each breakpoint, the following information can be specified:

- The linear address where the breakpoint is to occur.
- The length of the breakpoint location: 1, 2, 4, or 8 bytes (refer to the notes in Section 20.2.4).
- The operation that must be performed at the address for a debug exception to be generated.
- Whether the breakpoint is enabled.
- Whether the breakpoint condition was present when the debug exception was generated.

The following paragraphs describe the functions of flags and fields in the debug registers.

20.2.1 Debug Address Registers (DR0-DR3)

Each of the debug-address registers (DR0 through DR3) holds the 32-bit linear address of a breakpoint (see Figure 20-1). Breakpoint comparisons are made before physical address translation occurs. The contents of debug register DR7 further specifies breakpoint conditions.

20.2.2 Debug Registers DR4 and DR5

Debug registers DR4 and DR5 are reserved when debug extensions are enabled (when the DE flag in control register CR4 is set) and attempts to reference the DR4 and DR5 registers cause invalid-opcode exceptions (#UD). When debug extensions are not enabled (when the DE flag is clear), these registers are aliased to debug registers DR6 and DR7.

20.2.3 Debug Status Register (DR6)

The debug status register (DR6) reports debug conditions that were sampled at the time the last debug exception was generated (see Figure 20-1). Updates to this register only occur when an exception is generated. The flags in this register show the following information:

- **B0 through B3 (breakpoint condition detected) flags (bits 0 through 3)** — Indicates (when set) that its associated breakpoint condition was met when a debug exception was generated. These flags are set if the condition described for each breakpoint by the LEN_n and R/W_n flags in debug control register DR7 is true. They may or may not be set if the breakpoint is not enabled by the Ln or the Gn flags in register DR7. Therefore on a #DB, a debug handler should check only those B0-B3 bits which correspond to an enabled breakpoint.
- **BLD (bus-lock detected) flag (bit 11)** — Indicates (when **clear**) that the debug exception was triggered by the assertion of a bus lock when $CPL > 0$ and OS bus-lock detection was enabled (see Section 20.3.1.6). Other debug exceptions do not modify this bit. To avoid confusion in identifying debug exceptions, software debug-exception handlers should set bit 11 to 1 before returning. (Software that never enables OS bus-lock detection need not do this as $DR6[11] = 1$ following reset.) This bit is always 1 if the processor does not support OS bus-lock detection.
- **BD (debug register access detected) flag (bit 13)** — Indicates that the next instruction in the instruction stream accesses one of the debug registers (DR0 through DR7). This flag is enabled when the GD (general detect) flag in debug control register DR7 is set. See Section 20.2.4, “Debug Control Register (DR7),” for further explanation of the purpose of this flag.
- **BS (single step) flag (bit 14)** — Indicates (when set) that the debug exception was triggered by the single-step execution mode (enabled with the TF flag in the EFLAGS register). The single-step mode is the highest-priority debug exception. When the BS flag is set, any of the other debug status bits also may be set.
- **BT (task switch) flag (bit 15)** — Indicates (when set) that the debug exception resulted from a task switch where the T flag (debug trap flag) in the TSS of the target task was set. See Section 10.2.1, “Task-State Segment (TSS),” for the format of a TSS. There is no flag in debug control register DR7 to enable or disable this exception; the T flag of the TSS is the only enabling flag.
- **RTM (restricted transactional memory) flag (bit 16)** — Indicates (when **clear**) that a debug exception (#DB) or breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 20.3.3). This bit is set for any other debug exception (including all those that occur when advanced debugging of RTM transactional regions is not enabled). This bit is always 1 if the processor does not support RTM.

Certain debug exceptions may clear bits 0-3. The remaining contents of the DR6 register are never cleared by the processor. To avoid confusion in identifying debug exceptions, debug handlers should clear the register (except bit 16, which they should set) before returning to the interrupted task.

20.2.4 Debug Control Register (DR7)

The debug control register (DR7) enables or disables breakpoints and sets breakpoint conditions (see Figure 20-1). The flags and fields in this register control the following things:

- **L0 through L3 (local breakpoint enable) flags (bits 0, 2, 4, and 6)** — Enables (when set) the breakpoint condition for the associated breakpoint for the current task. When a breakpoint condition is detected and its associated *L_n* flag is set, a debug exception is generated. The processor automatically clears these flags on every task switch to avoid unwanted breakpoint conditions in the new task.
- **G0 through G3 (global breakpoint enable) flags (bits 1, 3, 5, and 7)** — Enables (when set) the breakpoint condition for the associated breakpoint for all tasks. When a breakpoint condition is detected and its associated *G_n* flag is set, a debug exception is generated. The processor does not clear these flags on a task switch, allowing a breakpoint to be enabled for all tasks.
- **LE and GE (local and global exact breakpoint enable) flags (bits 8, 9)** — This feature is not supported in the P6 family processors, later IA-32 processors, and Intel 64 processors. When set, these flags cause the processor to detect the exact instruction that caused a data breakpoint condition. For backward and forward compatibility with other Intel processors, we recommend that the LE and GE flags be set to 1 if exact breakpoints are required.
- **RTM (restricted transactional memory) flag (bit 11)** — Enables (when set) advanced debugging of RTM transactional regions (see Section 20.3.3). This advanced debugging is enabled only if IA32_DEBUGCTL.RTM is also set.
- **GD (general detect enable) flag (bit 13)** — Enables (when set) debug-register protection, which causes a debug exception to be generated prior to any MOV instruction that accesses a debug register. When such a condition is detected, the BD flag in debug status register DR6 is set prior to generating the exception. This condition is provided to support in-circuit emulators.

When the emulator needs to access the debug registers, emulator software can set the GD flag to prevent interference from the program currently executing on the processor.

The processor clears the GD flag upon entering to the debug exception handler, to allow the handler access to the debug registers.

- **R/W0 through R/W3 (read/write) fields (bits 16, 17, 20, 21, 24, 25, 28, and 29)** — Specifies the breakpoint condition for the corresponding breakpoint. The DE (debug extensions) flag in control register CR4 determines how the bits in the R/W_{*n*} fields are interpreted. When the DE flag is set, the processor interprets bits as follows:

- 00 — Break on instruction execution only.
- 01 — Break on data writes only.
- 10 — Break on I/O reads or writes.
- 11 — Break on data reads or writes but not instruction fetches.

When the DE flag is clear, the processor interprets the R/W_{*n*} bits the same as for the Intel386™ and Intel486™ processors, which is as follows:

- 00 — Break on instruction execution only.
- 01 — Break on data writes only.
- 10 — Undefined.
- 11 — Break on data reads or writes but not instruction fetches.

- **LEN0 through LEN3 (Length) fields (bits 18, 19, 22, 23, 26, 27, 30, and 31)** — Specify the size of the memory location at the address specified in the corresponding breakpoint address register (DR0 through DR3). These fields are interpreted as follows:

- 00 — 1-byte length.
- 01 — 2-byte length.
- 10 — Undefined (or 8 byte length, see note below).
- 11 — 4-byte length.

If the corresponding R/W_{*n*} field in register DR7 is 00 (instruction execution), then the LEN_{*n*} field should also be 00. The effect of using other lengths is undefined. See Section 20.2.5, “Breakpoint Field Recognition,” below.

NOTES

For Pentium® 4 and Intel® Xeon® processors with a CPUID signature corresponding to family 15 (model 3, 4, and 6), break point conditions permit specifying 8-byte length on data read/write with an of encoding 10B in the LEN_n field.

Encoding 10B is also supported in processors based on Intel Core microarchitecture or enhanced Intel Core microarchitecture, the respective CPUID signatures corresponding to family 6, model 15, and family 6, DisplayModel value 23 (see the CPUID instruction in Chapter 3, “Instruction Set Reference, A-L,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A). The Encoding 10B is supported in processors based on Intel Atom® microarchitecture, with CPUID signature of family 6, DisplayModel value 1CH. The encoding 10B is undefined for other processors.

20.2.5 Breakpoint Field Recognition

Breakpoint address registers (debug registers DR0 through DR3) and the LEN_n fields for each breakpoint define a range of sequential byte addresses for a data or I/O breakpoint. The LEN_n fields permit specification of a 1-, 2-, 4- or 8-byte range, beginning at the linear address specified in the corresponding debug register (DR_n). Two-byte ranges must be aligned on word boundaries; 4-byte ranges must be aligned on doubleword boundaries, 8-byte ranges must be aligned on quadword boundaries. I/O addresses are zero-extended (from 16 to 32 bits, for comparison with the breakpoint address in the selected debug register). These requirements are enforced by the processor; it uses LEN_n field bits to mask the lower address bits in the debug registers. Unaligned data or I/O breakpoint addresses do not yield valid results.

A data breakpoint for reading or writing data is triggered if any of the bytes participating in an access is within the range defined by a breakpoint address register and its LEN_n field. Table 20-1 provides an example setup of debug registers and data accesses that would subsequently trap or not trap on the breakpoints.

A data breakpoint for an unaligned operand can be constructed using two breakpoints, where each breakpoint is byte-aligned and the two breakpoints together cover the operand. The breakpoints generate exceptions only for the operand, not for neighboring bytes.

Instruction breakpoint addresses must have a length specification of 1 byte (the LEN_n field is set to 00). Instruction breakpoints for other operand sizes are undefined. The processor recognizes an instruction breakpoint address only when it points to the first byte of an instruction. If the instruction has prefixes, the breakpoint address must point to the first prefix.

Table 20-1. Breakpoint Examples

Debug Register Setup			
Debug Register	R/W _n	Breakpoint Address	LEN _n
DR0	R/W0 = 11 (Read/Write)	A0001H	LEN0 = 00 (1 byte)
DR1	R/W1 = 01 (Write)	A0002H	LEN1 = 00 (1 byte)
DR2	R/W2 = 11 (Read/Write)	B0002H	LEN2 = 01) (2 bytes)
DR3	R/W3 = 01 (Write)	C0000H	LEN3 = 11 (4 bytes)
Data Accesses			
Operation		Address	Access Length (In Bytes)
Data operations that trap			
- Read or write		A0001H	1
- Read or write		A0001H	2
- Write		A0002H	1
- Write		A0002H	2
- Read or write		B0001H	4
- Read or write		B0002H	1
- Read or write		B0002H	2
- Write		C0000H	4
- Write		C0001H	2
- Write		C0003H	1

Table 20-1. Breakpoint Examples (Contd.)

Debug Register Setup			
Debug Register	R/Wn	Breakpoint Address	LENn
Data operations that do not trap - Read or write - Read - Read or write - Read or write - Read - Read or write		A0000H	1
		A0002H	1
		A0003H	4
		B0000H	2
		C0000H	2
		C0004H	4

20.2.6 Debug Registers and Intel® 64 Processors

For Intel 64 architecture processors, debug registers DR0–DR7 are 64 bits. In 16-bit or 32-bit modes (protected mode and compatibility mode), writes to a debug register fill the upper 32 bits with zeros. Reads from a debug register return the lower 32 bits. In 64-bit mode, MOV DRn instructions read or write all 64 bits. Operand-size prefixes are ignored.

In 64-bit mode, the upper 32 bits of DR6 and DR7 are reserved and must be written with zeros. Writing 1 to any of the upper 32 bits results in a #GP(0) exception (see Figure 20-2). All 64 bits of DR0–DR3 are writable by software. However, MOV DRn instructions do not check that addresses written to DR0–DR3 are in the linear-address limits of the processor implementation (address matching is supported only on valid addresses generated by the processor implementation). Breakpoint conditions for 8-byte memory read/writes are supported in all modes.

20.3 DEBUG EXCEPTIONS

The Intel 64 and IA-32 architectures dedicate two interrupt vectors to handling debug exceptions: vector 1 (debug exception, #DB) and vector 3 (breakpoint exception, #BP). The following sections describe how these exceptions are generated and typical exception handler operations.

20.3.1 Debug Exception (#DB)—Interrupt Vector 1

The debug-exception handler is usually a debugger program or part of a larger software system. The processor generates a debug exception for any of several conditions. The debugger checks flags in the DR6 and DR7 registers to determine which condition caused the exception and which other conditions might apply. Table 20-2 shows the states of these flags following the generation of each kind of breakpoint condition.

Instruction-breakpoint and general-detect condition (see Section 20.3.1.3, “General-Detect Exception Condition”) result in faults; other debug-exception conditions result in traps. The debug exception may report one or both at one time. The following sections describe each class of debug exception.

The INT1 instruction generates a debug exception as a trap. Hardware vendors may use the INT1 instruction for hardware debug. For that reason, Intel recommends software vendors instead use the INT3 instruction for software breakpoints.

See also: Chapter 7, “Interrupt 1—Debug Exception (#DB),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

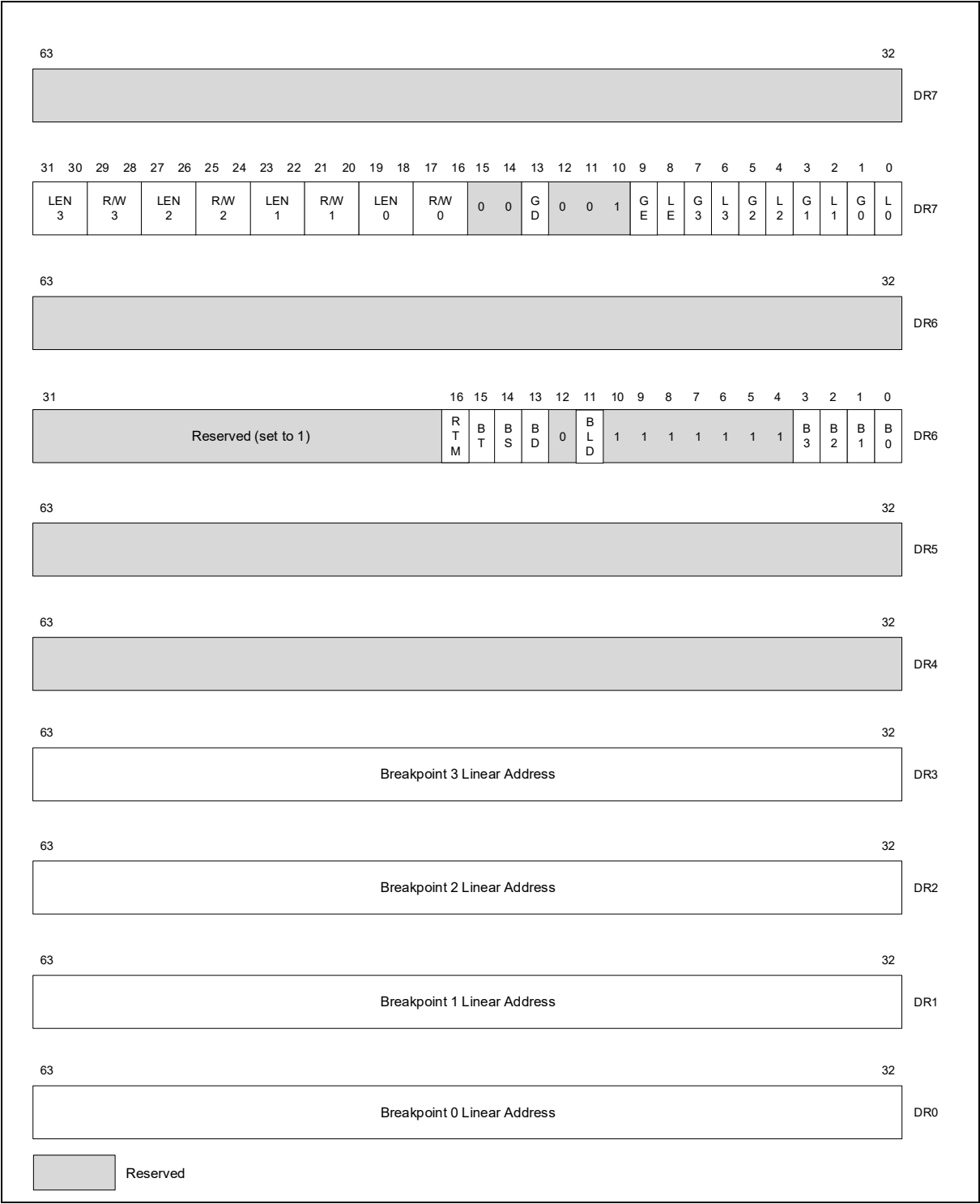


Figure 20-2. DR6/DR7 Layout on Processors Supporting Intel® 64 Architecture

Table 20-2. Debug Exception Conditions

Debug or Breakpoint Condition	DR6 Flags Tested	DR7 Flags Tested	Exception Class
Single-step trap	BS = 1		Trap
Instruction breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 0	Fault
Data write breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 1	Trap
I/O read or write breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 2	Trap
Data read or write (but not instruction fetches), at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 3	Trap
General detect fault, resulting from an attempt to modify debug registers (usually in conjunction with in-circuit emulation)	BD = 1	None	Fault
Task switch	BT = 1	None	Trap
INT1 instruction	None	None	Trap

20.3.1.1 Instruction-Breakpoint Exception Condition

The processor reports an instruction breakpoint when it attempts to execute an instruction at an address specified in a breakpoint-address register (DR0 through DR3) that has been set up to detect instruction execution (R/W flag is set to 0). Upon reporting the instruction breakpoint, the processor generates a fault-class, debug exception (#DB) before it executes the target instruction for the breakpoint.

Instruction breakpoints are the highest priority debug exceptions. They are serviced before any other exceptions detected during the decoding or execution of an instruction. However, if an instruction breakpoint is placed on an instruction located immediately after a POP SS/MOV SS instruction, the breakpoint will be suppressed as if EFLAGS.RF were 1 (see the next paragraph and Section 7.8.3, “Masking Exceptions and Interrupts When Switching Stacks,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A).

Because the debug exception for an instruction breakpoint is generated before the instruction is executed, if the instruction breakpoint is not removed by the exception handler; the processor will detect the instruction breakpoint again when the instruction is restarted and generate another debug exception. To prevent looping on an instruction breakpoint, the Intel 64 and IA-32 architectures provide the RF flag (resume flag) in the EFLAGS register (see Section 2.3, “System Flags and Fields in the EFLAGS Register,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). When the RF flag is set, the processor ignores instruction breakpoints.

All Intel 64 and IA-32 processors manage the RF flag as follows. The RF Flag is cleared at the start of the instruction after the check for instruction breakpoints, CS limit violations, and FP exceptions. Task Switches and IRETD/IRETQ instructions transfer the RF image from the TSS/stack to the EFLAGS register.

When calling an event handler, Intel 64 and IA-32 processors establish the value of the RF flag in the EFLAGS image pushed on the stack:

- For any fault-class exception except a debug exception generated in response to an instruction breakpoint, the value pushed for RF is 1.
- For any interrupt arriving after any iteration of a repeated string instruction but the last iteration, the value pushed for RF is 1.
- For any trap-class exception generated by any iteration of a repeated string instruction but the last iteration, the value pushed for RF is 1.
- For other cases, the value pushed for RF is the value that was in EFLAG.RF at the time the event handler was called. This includes:
 - Debug exceptions generated in response to instruction breakpoints
 - Hardware-generated interrupts arriving between instructions (including those arriving after the last iteration of a repeated string instruction)

- Trap-class exceptions generated after an instruction completes (including those generated after the last iteration of a repeated string instruction)
- Software-generated interrupts (RF is pushed as 0, since it was cleared at the start of the software interrupt)

As noted above, the processor does not set the RF flag prior to calling the debug exception handler for debug exceptions resulting from instruction breakpoints. The debug exception handler can prevent recurrence of the instruction breakpoint by setting the RF flag in the EFLAGS image on the stack. If the RF flag in the EFLAGS image is set when the processor returns from the exception handler, it is copied into the RF flag in the EFLAGS register by IRETD/IRETQ or a task switch that causes the return. The processor then ignores instruction breakpoints for the duration of the next instruction. (Note that the POPF, POPFD, and IRET instructions do not transfer the RF image into the EFLAGS register.) Setting the RF flag does not prevent other types of debug-exception conditions (such as, I/O or data breakpoints) from being detected, nor does it prevent non-debug exceptions from being generated.

For the Pentium processor, when an instruction breakpoint coincides with another fault-type exception (such as a page fault), the processor may generate one spurious debug exception after the second exception has been handled, even though the debug exception handler set the RF flag in the EFLAGS image. To prevent a spurious exception with Pentium processors, all fault-class exception handlers should set the RF flag in the EFLAGS image.

20.3.1.2 Data Memory and I/O Breakpoint Exception Conditions

Data memory and I/O breakpoints are reported when the processor attempts to access a memory or I/O address specified in a breakpoint-address register (DR0 through DR3) that has been set up to detect data or I/O accesses (R/W flag is set to 1, 2, or 3). The processor generates the exception after it executes the instruction that made the access, so these breakpoint condition causes a trap-class exception to be generated.

Because data breakpoints are traps, an instruction that writes memory overwrites the original data before the debug exception generated by a data breakpoint is generated. If a debugger needs to save the contents of a write breakpoint location, it should save the original contents before setting the breakpoint. The handler can report the saved value after the breakpoint is triggered. The address in the debug registers can be used to locate the new value stored by the instruction that triggered the breakpoint.

If a data breakpoint is detected during an iteration of a string instruction executed with fast-string operation (see Section 7.3.9.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1), delivery of the resulting debug exception may be delayed until completion of the corresponding group of iterations.

Intel486 and later processors ignore the GE and LE flags in DR7. In Intel386 processors, exact data breakpoint matching does not occur unless it is enabled by setting the LE and/or the GE flags.

For repeated INS and OUTS instructions that generate an I/O-breakpoint debug exception, the processor generates the exception after the completion of the first iteration. Repeated INS and OUTS instructions generate a data-breakpoint debug exception after the iteration in which the memory address breakpoint location is accessed.

If an execution of the MOV or POP instruction loads the SS register and encounters a data breakpoint, the resulting debug exception is delivered after completion of the next instruction (the one after the MOV or POP).

Any pending data or I/O breakpoints are lost upon delivery of an exception. For example, if a machine-check exception (#MC) occurs following an instruction that encounters a data breakpoint (but before the resulting debug exception is delivered), the data breakpoint is lost. If a MOV or POP instruction that loads the SS register encounters a data breakpoint, the data breakpoint is lost if the next instruction causes a fault.

Delivery of events due to INT n , INT3, or INTO does not cause a loss of data breakpoints. If a MOV or POP instruction that loads the SS register encounters a data breakpoint, and the next instruction is software interrupt (INT n , INT3, or INTO), a debug exception (#DB) resulting from a data breakpoint will be delivered after the transition to the software-interrupt handler. The #DB handler should account for the fact that the #DB may have been delivered after a invocation of a software-interrupt handler, and in particular that the CPL may have changed between recognition of the data breakpoint and delivery of the #DB.

20.3.1.3 General-Detect Exception Condition

When the GD flag in DR7 is set, the general-detect debug exception occurs when a program attempts to access any of the debug registers (DR0 through DR7) at the same time they are being used by another application, such as an emulator or debugger. This protection feature guarantees full control over the debug registers when required. The

debug exception handler can detect this condition by checking the state of the BD flag in the DR6 register. The processor generates the exception before it executes the MOV instruction that accesses a debug register, which causes a fault-class exception to be generated.

20.3.1.4 Single-Step Exception Condition

The processor generates a single-step debug exception if (while an instruction is being executed) it detects that the TF flag in the EFLAGS register is set. The exception is a trap-class exception, because the exception is generated after the instruction is executed. The processor will not generate this exception after the instruction that sets the TF flag. For example, if the POPF instruction is used to set the TF flag, a single-step trap does not occur until after the instruction that follows the POPF instruction.

The processor clears the TF flag before calling the exception handler. If the TF flag was set in a TSS at the time of a task switch, the exception occurs after the first instruction is executed in the new task.

The TF flag normally is not cleared by privilege changes inside a task. The INT *n*, INT3, and INTO instructions, however, do clear this flag. Therefore, software debuggers that single-step code must recognize and emulate INT *n* or INTO instructions rather than executing them directly. To maintain protection, the operating system should check the CPL after any single-step trap to see if single stepping should continue at the current privilege level.

The interrupt priorities guarantee that, if an external interrupt occurs, single stepping stops. When both an external interrupt and a single-step interrupt occur together, the single-step interrupt is processed first. This operation clears the TF flag. After saving the return address or switching tasks, the external interrupt input is examined before the first instruction of the single-step handler executes. If the external interrupt is still pending, then it is serviced. The external interrupt handler does not run in single-step mode. To single step an interrupt handler, single step an INT *n* instruction that calls the interrupt handler.

If an occurrence of the MOV or POP instruction loads the SS register executes with EFLAGS.TF = 1, no single-step debug exception occurs following the MOV or POP instruction.

20.3.1.5 Task-Switch Exception Condition

The processor generates a debug exception after a task switch if the T flag of the new task's TSS is set. This exception is generated after program control has passed to the new task, and prior to the execution of the first instruction of that task. The exception handler can detect this condition by examining the BT flag of the DR6 register.

If entry 1 (#DB) in the IDT is a task gate, the T bit of the corresponding TSS should not be set. Failure to observe this rule will put the processor in a loop.

20.3.1.6 OS Bus-Lock Detection

OS bus-lock detection is a feature that causes the processor to generate a debug exception (called a **bus-lock detection debug exception**) if it detects that a bus lock has been asserted (see Section 11.1.2). Such an exception is a trap-class exception, because it is generated after execution of an instruction that asserts a bus lock. The exception thus does not prevent assertion of the bus lock. Delivery of a bus-lock detection debug exception clears DR6.BLD.

Software can enable OS bus-lock detection by setting IA32_DEBUGCTL.BLD[bit 2]. Bus-lock detection debug exceptions occur only if CPL > 0.

20.3.2 Breakpoint Exception (#BP)—Interrupt Vector 3

The breakpoint exception (interrupt 3) is caused by execution of an INT3 instruction. See Chapter 7, "Interrupt 3—Breakpoint Exception (#BP)." Debuggers use breakpoint exceptions in the same way that they use the breakpoint registers; that is, as a mechanism for suspending program execution to examine registers and memory locations. With earlier IA-32 processors, breakpoint exceptions are used extensively for setting instruction breakpoints.

With the Intel386 and later IA-32 processors, it is more convenient to set breakpoints with the breakpoint-address registers (DR0 through DR3). However, the breakpoint exception still is useful for breakpointing debuggers,

because a breakpoint exception can call a separate exception handler. The breakpoint exception is also useful when it is necessary to set more breakpoints than there are debug registers or when breakpoints are being placed in the source code of a program under development.

20.3.3 Debug Exceptions, Breakpoint Exceptions, and Restricted Transactional Memory (RTM)

Chapter 16, “Programming with Intel® AVX10,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, describes Restricted Transactional Memory (RTM). This is an instruction-set interface that allows software to identify **transactional regions** (or critical sections) using the XBEGIN and XEND instructions.

Execution of an RTM transactional region begins with an XBEGIN instruction. If execution of the region successfully reaches an XEND instruction, the processor ensures that all memory operations performed within the region appear to have occurred instantaneously when viewed from other logical processors. Execution of an RTM transaction region does not succeed if the processor cannot commit the updates atomically. When this happens, the processor rolls back the execution, a process referred to as a **transactional abort**. In this case, the processor discards all updates performed in the region, restores architectural state to appear as if the execution had not occurred, and resumes execution at a fallback instruction address that was specified with the XBEGIN instruction.

If debug exception (#DB) or breakpoint exception (#BP) occurs within an RTM transaction region, a transactional abort occurs, the processor sets EAX[4], and no exception is delivered.

Software can enable **advanced debugging of RTM transactional regions** by setting DR7.RTM[bit 11] and IA32_DEBUGCTL.RTM[bit 15]. If these bits are both set, the transactional abort caused by a #DB or #BP within an RTM transaction region does **not** resume execution at the fallback instruction address specified with the XBEGIN instruction that begin the region. Instead, execution is resumed at that XBEGIN instruction, and a #DB is delivered. (A #DB is delivered even if the transactional abort was caused by a #BP.) Such a #DB will clear DR6.RTM[bit 16] (all other debug exceptions set DR6[16]).

20.4 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING OVERVIEW

P6 family processors introduced the ability to set breakpoints on taken branches, interrupts, and exceptions, and to single-step from one branch to the next. This capability has been modified and extended in the Pentium 4, Intel Xeon, Pentium M, Intel® Core™ Solo, Intel® Core™ Duo, Intel® Core™2 Duo, Intel® Core™ i7 and Intel Atom® processors to allow logging of branch trace messages in a branch trace store (BTS) buffer in memory.

See the following sections for processor specific implementation of last branch, interrupt, and exception recording:

- Section 20.5, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors).”
- Section 20.6, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Microarchitecture.”
- Section 20.9, “Last Branch, Interrupt, and Exception Recording for Processors based on Nehalem Microarchitecture.”
- Section 20.10, “Last Branch, Interrupt, and Exception Recording for Processors based on Sandy Bridge Microarchitecture.”
- Section 20.11, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Haswell Microarchitecture.”
- Section 20.12, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture.”
- Section 20.14, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ Solo and Intel® Core™ Duo Processors).”
- Section 20.15, “Last Branch, Interrupt, and Exception Recording (Pentium M Processors).”
- Section 20.16, “Last Branch, Interrupt, and Exception Recording (P6 Family Processors).”

The following subsections of Section 20.4 describe common features of profiling branches. These features are generally enabled using the IA32_DEBUGCTL MSR (older processor may have implemented a subset or model-specific features, see definitions of MSR_DEBUGCTLA, MSR_DEBUGCTLB, MSR_DEBUGCTL).

20.4.1 IA32_DEBUGCTL MSR

The **IA32_DEBUGCTL** MSR provides bit field controls to enable debug trace interrupts, debug trace stores, trace messages enable, single stepping on branches, last branch record recording, and to control freezing of LBR stack or performance counters on a PMI request. IA32_DEBUGCTL MSR is located at register address 01D9H.

See Figure 20-3 for the MSR layout and the bullets below for a description of the flags:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the Section 20.5.1, “LBR Stack” (Intel® Core™2 Duo and Intel Atom® processor family) and Section 20.9.1, “LBR Stack” (processors based on Nehalem microarchitecture).
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 20.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
- **BLD (bus-lock detection) flag (bit 2)** — If this bit is set, OS bus-lock detection is enabled when CPL > 0. See Section 20.3.1.6.
- **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception; it sends the branch record out on the system bus as a branch trace message (BTM). See Section 20.4.4, “Branch Trace Messages,” for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7)** — When set, the flag enables BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 20.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bit 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 20.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

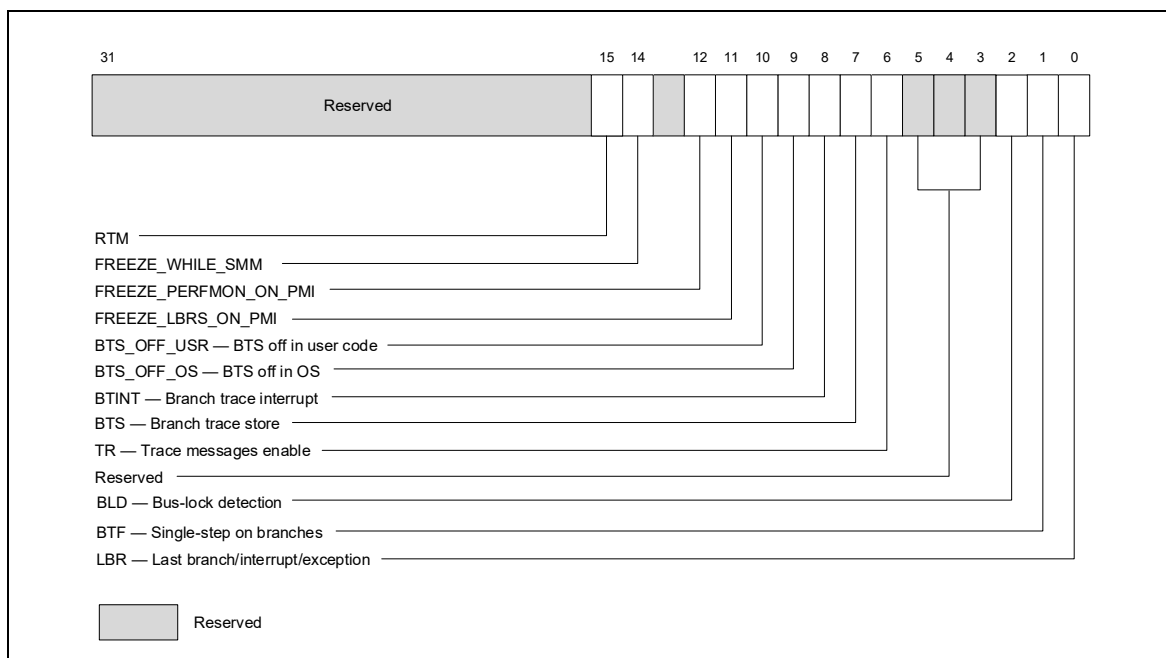


Figure 20-3. IA32_DEBUGCTL MSR for Processors Based on Intel® Core™ Microarchitecture

- **BTS_OFF_OS (branch trace off in privileged code) flag (bit 9)** — When set, BTS or BTM is skipped if CPL is 0. See Section 20.13.2.
- **BTS_OFF_USR (branch trace off in user code) flag (bit 10)** — When set, BTS or BTM is skipped if CPL is greater than 0. See Section 20.13.2.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — When set, the LBR stack is frozen on a hardware PMI request (e.g., when a counter overflows and is configured to trigger PMI). See Section 20.4.7 for details.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — When set, the performance counters (IA32_PMCx and IA32_FIXED_CTRx) are frozen on a PMI request. See Section 20.4.7 for details.
- **FREEZE_WHILE_SMM (bit 14)** — If this bit is set, upon the delivery of an SMI, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTF, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler. If Intel Thread Director support was enabled before transferring control to the SMI handler, then the processor will also reset the Intel Thread Director history (see Section 17.6.11 for more details about Intel Thread Director enable, reset, and history reset operations).
Subsequently, the enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its service. If Intel Thread Director support is enabled when RSM is executed, then the processor resets the Intel Thread Director history.
Note that system software must check if the processor supports the IA32_DEBUGCTL.FREEZE_WHILE_SMM control bit. IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 22.8 for details of detecting the presence of IA32_PERF_CAPABILITIES MSR.
- **RTM (bit 15)** — If this bit is set, advanced debugging of RTM transactional regions is enabled if DR7.RTM is also set. See Section 20.3.3.

20.4.2 Monitoring Branches, Exceptions, and Interrupts

When the LBR flag (bit 0) in the IA32_DEBUGCTL MSR is set, the processor automatically begins recording branch records for taken branches, interrupts, and exceptions (except for debug exceptions) in the LBR stack MSRs.

When the processor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler. This action does not clear previously stored LBR stack MSRs.

A debugger can use the linear addresses in the LBR stack to re-set breakpoints in the breakpoint address registers (DR0 through DR3). This allows a backward trace from the manifestation of a particular bug toward its source.

On some processors, if the LBR flag is cleared and TR flag in the IA32_DEBUGCTL MSR remains set, the processor will continue to update LBR stack MSRs. This is because those processors use the entries in the LBR stack in the process of generating BTM/BTS records. A #DB does not automatically clear the TR flag.

20.4.3 Single-Stepping on Branches

When software sets both the BTF flag (bit 1) in the IA32_DEBUGCTL MSR and the TF flag in the EFLAGS register, the processor generates a single-step debug exception only after instructions that cause a branch.¹ This mechanism allows a debugger to single-step on control transfers caused by branches. This “branch single stepping” helps isolate a bug to a particular block of code before instruction single-stepping further narrows the search. The processor clears the BTF flag when it generates a debug exception. The debugger must set the BTF flag before resuming program execution to continue single-stepping on branches.

1. Executions of CALL, IRET, and JMP that cause task switches never cause single-step debug exceptions (regardless of the value of the BTF flag). A debugger desiring debug exceptions on switches to a task should set the T flag (debug trap flag) in the TSS of that task. See Section 10.2.1, “Task-State Segment (TSS).”

20.4.4 Branch Trace Messages

Setting the TR flag (bit 6) in the IA32_DEBUGCTL MSR enables branch trace messages (BTMs). Thereafter, when the processor detects a branch, exception, or interrupt, it sends a branch record out on the system bus as a BTM. A debugging device that is monitoring the system bus can read these messages and synchronize operations with taken branch, interrupt, and exception events.

When interrupts or exceptions occur in conjunction with a taken branch, additional BTMs are sent out on the bus, as described in Section 20.4.2, “Monitoring Branches, Exceptions, and Interrupts.”

For the P6 processor family, Pentium M processor family, and processors based on Intel Core microarchitecture, TR and LBR bits can not be set at the same time due to hardware limitation. The content of LBR stack is undefined when TR is set.

For processors with Intel NetBurst microarchitecture, Intel Atom processors, and Intel Core and related Intel Xeon processors both starting with the Nehalem microarchitecture, the processor can collect branch records in the LBR stack and at the same time send/store BTMs when both the TR and LBR flags are set in the IA32_DEBUGCTL MSR (or the equivalent MSR_DEBUGCTLA, MSR_DEBUGCTLB).

The following exception applies:

- BTM may not be observable on Intel Atom processor families that do not provide an externally visible system bus (i.e., processors based on the Silvermont microarchitecture or later).

20.4.4.1 Branch Trace Message Visibility

Branch trace message (BTM) visibility is implementation specific and limited to systems with a front side bus (FSB). BTMs may not be visible to newer system link interfaces or a system bus that deviates from a traditional FSB.

20.4.5 Branch Trace Store (BTS)

A trace of taken branches, interrupts, and exceptions is useful for debugging code by providing a method of determining the decision path taken to reach a particular code location. The LBR flag (bit 0) of IA32_DEBUGCTL provides a mechanism for capturing records of taken branches, interrupts, and exceptions and saving them in the last branch record (LBR) stack MSRs, setting the TR flag for sending them out onto the system bus as BTMs. The branch trace store (BTS) mechanism provides the additional capability of saving the branch records in a memory-resident BTS buffer, which is part of the DS save area. The BTS buffer can be configured to be circular so that the most recent branch records are always available or it can be configured to generate an interrupt when the buffer is nearly full so that all the branch records can be saved. The BTINT flag (bit 8) can be used to enable the generation of interrupt when the BTS buffer is full. See Section 20.4.9.2, “Setting Up the DS Save Area,” for additional details.

Setting this flag (BTS) alone can greatly reduce the performance of the processor. CPL-qualified branch trace storing mechanism can help mitigate the performance impact of sending/logging branch trace messages.

20.4.6 CPL-Qualified Branch Trace Mechanism

CPL-qualified branch trace mechanism is available to a subset of Intel 64 and IA-32 processors that support the branch trace storing mechanism. The processor supports the CPL-qualified branch trace mechanism if CPUID.01H:ECX[4] = 1.

The CPL-qualified branch trace mechanism is described in Section 20.4.9.4. System software can selectively specify CPL qualification to not send/store Branch Trace Messages associated with a specified privilege level. Two bit fields, BTS_OFF_USR (bit 10) and BTS_OFF_OS (bit 9), are provided in the debug control register to specify the CPL of BTMs that will not be logged in the BTS buffer or sent on the bus.

20.4.7 Freezing LBR and Performance Counters on PMI

Many issues may generate a performance monitoring interrupt (PMI); a PMI service handler will need to determine cause to handle the situation. Two capabilities that allow a PMI service routine to improve branch tracing and performance monitoring are available for processors supporting architectural performance monitoring version 2 or

greater (i.e., CPUID.0AH:EAX[7:0] > 1). These capabilities provides the following interface in IA32_DEBUGCTL to reduce runtime overhead of PMI servicing, profiler-contributed skew effects on analysis or counter metrics:

- **Freezing LBRs on PMI (bit 11)** — Allows the PMI service routine to ensure the content in the LBR stack are associated with the target workload and not polluted by the branch flows of handling the PMI. Depending on the version ID enumerated by CPUID.0AH:EAX.VERSION[7:0], two flavors are supported:
 - Legacy Freeze_LBR_on_PMI is supported for ArchPerfMonVerID <= 3 and ArchPerfMonVerID > 1. If IA32_DEBUGCTL.Freeze_LBR_On_PMI = 1, the LBR is frozen on the overflowed condition of the buffer area, the processor clears the LBR bit (bit 0) in IA32_DEBUGCTL. Software must then re-enable IA32_DEBUGCTL.LBR to resume recording branches. When using this feature, software should be careful about writes to IA32_DEBUGCTL to avoid re-enabling LBRs by accident if they were just disabled.
 - Streamlined Freeze_LBR_on_PMI is supported for ArchPerfMonVerID >= 4. If IA32_DEBUGCTL.Freeze_LBR_On_PMI = 1, the processor behaves as follows:
 - sets IA32_PERF_GLOBAL_STATUS.LBR_Frz = 1 to disable recording, but does not change the LBR bit (bit 0) in IA32_DEBUGCTL. The LBRs are frozen on the overflowed condition of the buffer area.
- **Freezing PMCs on PMI (bit 12)** — Allows the PMI service routine to ensure the content in the performance counters are associated with the target workload and not polluted by the PMI and activities within the PMI service routine. Depending on the version ID enumerated by CPUID.0AH:EAX.VERSION[7:0], two flavors are supported:
 - Legacy Freeze_PerfMon_on_PMI is supported for ArchPerfMonVerID <= 3 and ArchPerfMonVerID > 1. If IA32_DEBUGCTL.Freeze_PerfMon_On_PMI = 1, the performance counters are frozen on the counter overflowed condition when the processor clears the IA32_PERF_GLOBAL_CTRL MSR (see Figure 22-3). The PMCs affected include both general-purpose counters and fixed-function counters (see Section 22.6.2.1, “Fixed-function Performance Counters”). Software must re-enable counts by writing 1s to the corresponding enable bits in IA32_PERF_GLOBAL_CTRL before leaving a PMI service routine to continue counter operation.
 - Streamlined Freeze_PerfMon_on_PMI is supported for ArchPerfMonVerID >= 4. The processor behaves as follows:
 - sets IA32_PERF_GLOBAL_STATUS.CTR_Frz = 1 to disable counting on a counter overflow condition, but does not change the IA32_PERF_GLOBAL_CTRL MSR.

Freezing LBRs and PMCs on PMIs (both legacy and streamlined operation) occur when one of the following applies:

- A performance counter had an overflow and was programmed to signal a PMI in case of an overflow.
 - For the general-purpose counters; enabling PMI is done by setting bit 20 of the IA32_PERFEVTSELx register.
 - For the fixed-function counters; enabling PMI is done by setting the 3rd bit in the corresponding 4-bit control field of the MSR_PERF_FIXED_CTR_CTRL register (see Figure 22-1) or IA32_FIXED_CTR_CTRL MSR (see Figure 22-2).
- The PEBS buffer is almost full and reaches the interrupt threshold.
- The BTS buffer is almost full and reaches the interrupt threshold.

Table 20-3 compares the interaction of the processor with the PMI handler using the legacy versus streamlined Freeze_PerfMon_On_PMI interface.

Table 20-3. Legacy and Streamlined Operation with Freeze_PerfMon_On_PMI = 1, Counter Overflowed

Legacy Freeze_PerfMon_On_PMI	Streamlined Freeze_PerfMon_On_PMI	Comment
Processor freezes the counters on overflow	Processor freezes the counters on overflow	Unchanged
Processor clears IA32_PERF_GLOBAL_CTRL	Processor set IA32_PERF_GLOBAL_STATUS.CTR_FTZ	
Handler reads IA32_PERF_GLOBAL_STATUS(0x38E) to examine which counter(s) overflowed	mask = RDMSR(0x38E)	Similar
Handler services the PMI	Handler services the PMI	Unchanged
Handler writes 1s to IA32_PERF_GLOBAL_OVF_CTL (0x390)	Handler writes mask into IA32_PERF_GLOBAL_OVF_RESET (0x390)	
Processor clears IA32_PERF_GLOBAL_STATUS	Processor clears IA32_PERF_GLOBAL_STATUS	Unchanged
Handler re-enables IA32_PERF_GLOBAL_CTRL	None	Reduced software overhead

20.4.8 LBR Stack

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported across Intel 64 and IA-32 processor families. However, the number of MSRs in the LBR stack and the valid range of TOS pointer value can vary between different processor families. Table 20-4 lists the LBR stack size and TOS pointer range for several processor families according to the CPUID signatures of DisplayFamily_DisplayModel encoding (see the CPUID instruction in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A).

Table 20-4. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Component of an LBR Entry	Range of TOS Pointer
06_5CH, 06_5FH	32	FROM_IP, TO_IP	0 to 31
06_4EH, 06_5EH, 06_8EH, 06_9EH, 06_55H, 06_66H, 06_7AH, 06_67H, 06_6AH, 06_6CH, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_A5H, 06_A6H, 06_A7H, 06_A8H, 06_86H, 06_8AH, 06_96H, 06_9CH	32	FROM_IP, TO_IP, LBR_INFO ¹	0 to 31
06_3DH, 06_47H, 06_4FH, 06_56H, 06_3CH, 06_45H, 06_46H, 06_3FH, 06_2AH, 06_2DH, 06_3AH, 06_3EH, 06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	16	FROM_IP, TO_IP	0 to 15
06_17H, 06_1DH, 06_0FH	4	FROM_IP, TO_IP	0 to 3
06_37H, 06_4AH, 06_4CH, 06_4DH, 06_5AH, 06_5DH, 06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	8	FROM_IP, TO_IP	0 to 7

NOTES:

1. See Section 20.12.

The last branch recording mechanism tracks not only branch instructions (e.g., JMP, Jcc, LOOP, and CALL instructions), but also other operations that cause a change in the instruction pointer (e.g., external interrupts, traps, and faults). The branch recording mechanisms generally employs a set of MSRs, referred to as last branch record (LBR) stack. The size and exact locations of the LBR stack are generally model-specific (see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for model-specific MSR addresses).

- **Last Branch Record (LBR) Stack** — The LBR consists of N pairs of MSRs (N is listed in the LBR stack size column of Table 20-4) that store source and destination address of recent branches (see Figure 20-3):
 - MSR_LASTBRANCH_0_FROM_IP (address is model specific) through the next consecutive (N-1) MSR address store source addresses.
 - MSR_LASTBRANCH_0_TO_IP (address is model specific) through the next consecutive (N-1) MSR address store destination addresses.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant M bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address is model specific) contains an M-bit pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. The valid range of the M-bit POS pointer is given in Table 20-4.

20.4.8.1 LBR Stack and Intel® 64 Processors

LBR MSRs are 64-bits. In 64-bit mode, last branch records store the full address. Outside of 64-bit mode, the upper 32-bits of branch addresses will be stored as 0.

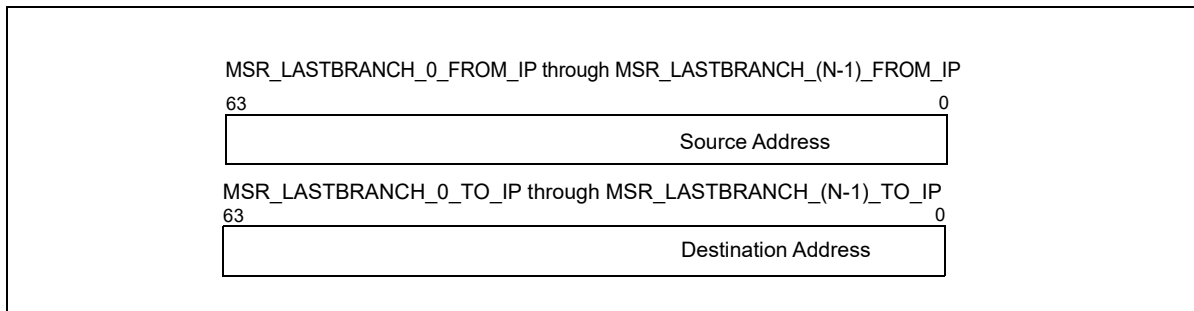


Figure 20-4. 64-bit Address Layout of LBR MSR

Software should query an architectural MSR IA32_PERF_CAPABILITIES[5:0] about the format of the address that is stored in the LBR stack. Four formats are defined by the following encoding:

- **000000B (32-bit record format)** — Stores 32-bit offset in current CS of respective source/destination,
- **000001B (64-bit LIP record format)** — Stores 64-bit linear address of respective source/destination,
- **000010B (64-bit EIP record format)** — Stores 64-bit offset (effective address) of respective source/destination.
- **000011B (64-bit EIP record format) and Flags** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction info is reported in the upper bit of 'FROM' registers in the LBR stack. See LBR stack details below for flag support and definition.
- **000100B (64-bit EIP record format), Flags, and TSX** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction and TSX info are reported in the upper bits of 'FROM' registers in the LBR stack.
- **000101B (64-bit EIP record format), Flags, TSX, and LBR_INFO** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction, TSX, and elapsed cycles since the last LBR update are reported in the LBR_INFO MSR stack.
- **000110B (64-bit LIP record format), Flags, and Cycles** — Stores 64-bit linear address (CS.Base + effective address) of respective source/destination. Misprediction info is reported in the upper bits of

'FROM' registers in the LBR stack. Elapsed cycles since the last LBR update are reported in the upper 16 bits of the 'TO' registers in the LBR stack (see Section 20.6).

- **000111B (64-bit LIP record format), Flags, and LBR_INFO** — Stores 64-bit linear address (CS.Base + effective address) of respective source/destination. Misprediction, and elapsed cycles since the last LBR update are reported in the LBR_INFO MSR stack.

Processor's support for the architectural MSR IA32_PERF_CAPABILITIES is provided by CPUID.01H:ECX.PERF_CAPABILITIES (bit 15).

20.4.8.2 LBR Stack and IA-32 Processors

The LBR MSRs in IA-32 processors introduced prior to Intel 64 architecture store the 32-bit "To Linear Address" and "From Linear Address" using the high and low half of each 64-bit MSR.

20.4.8.3 Last Exception Records and Intel 64 Architecture

Intel 64 and IA-32 processors also provide MSRs that store the branch record for the last branch taken prior to an exception or an interrupt. The location of the last exception record (LER) MSRs are model specific. The MSRs that store last exception records are 64-bits. If IA-32e mode is disabled, only the lower 32-bits of the address is recorded. If IA-32e mode is enabled, the processor writes 64-bit values into the MSR. In 64-bit mode, last exception records store 64-bit addresses; in compatibility mode, the upper 32-bits of last exception records are cleared.

20.4.9 BTS and DS Save Area

The **Debug store (DS)** feature flag (bit 21), returned by CPUID.01H:EDX[21] indicates that the processor provides the debug store (DS) mechanism. The DS mechanism allows:

- BTMs to be stored in a memory-resident BTS buffer. See Section 20.4.5, "Branch Trace Store (BTS)."
- Processor event-based sampling (PEBS) also uses the DS save area provided by debug store mechanism. The capability of PEBS varies across different microarchitectures. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)," and the relevant PEBS sub-sections across the core PMU sections in Chapter 22, "Performance Monitoring."

When CPUID.01H:EDX[21] is set:

- The BTS_UNAVAILABLE and PEBS_UNAVAILABLE flags in the IA32_MISC_ENABLE MSR indicate (when clear) the availability of the BTS and PEBS facilities, including the ability to set the BTS and BTINT bits in the appropriate DEBUGCTL MSR.
- The IA32_DS_AREA MSR exists and points to the DS save area.

The debug store (DS) save area is a software-designated area of memory that is used to collect the following two types of information:

- **Branch records** — When the BTS flag in the IA32_DEBUGCTL MSR is set, a branch record is stored in the BTS buffer in the DS save area whenever a taken branch, interrupt, or exception is detected.
- **PEBS records** — When a performance counter is configured for PEBS, a PEBS record is stored in the PEBS buffer in the DS save area after the counter overflow occurs. This record contains the architectural state of the processor (state of the 8 general purpose registers, EIP register, and EFLAGS register) at the next occurrence of the PEBS event that caused the counter to overflow. When the state information has been logged, the counter is automatically reset to a specified value, and event counting begins again. The content layout of a PEBS record varies across different implementations that support PEBS. See Section 22.6.2.4.2 for details of enumerating PEBS record format.

NOTES

Prior to processors based on the Goldmont microarchitecture, PEBS facility only supports a subset of implementation-specific precise events. See Section 22.5.3.1 for a PEBS enhancement that can generate records for both precise and non-precise events.

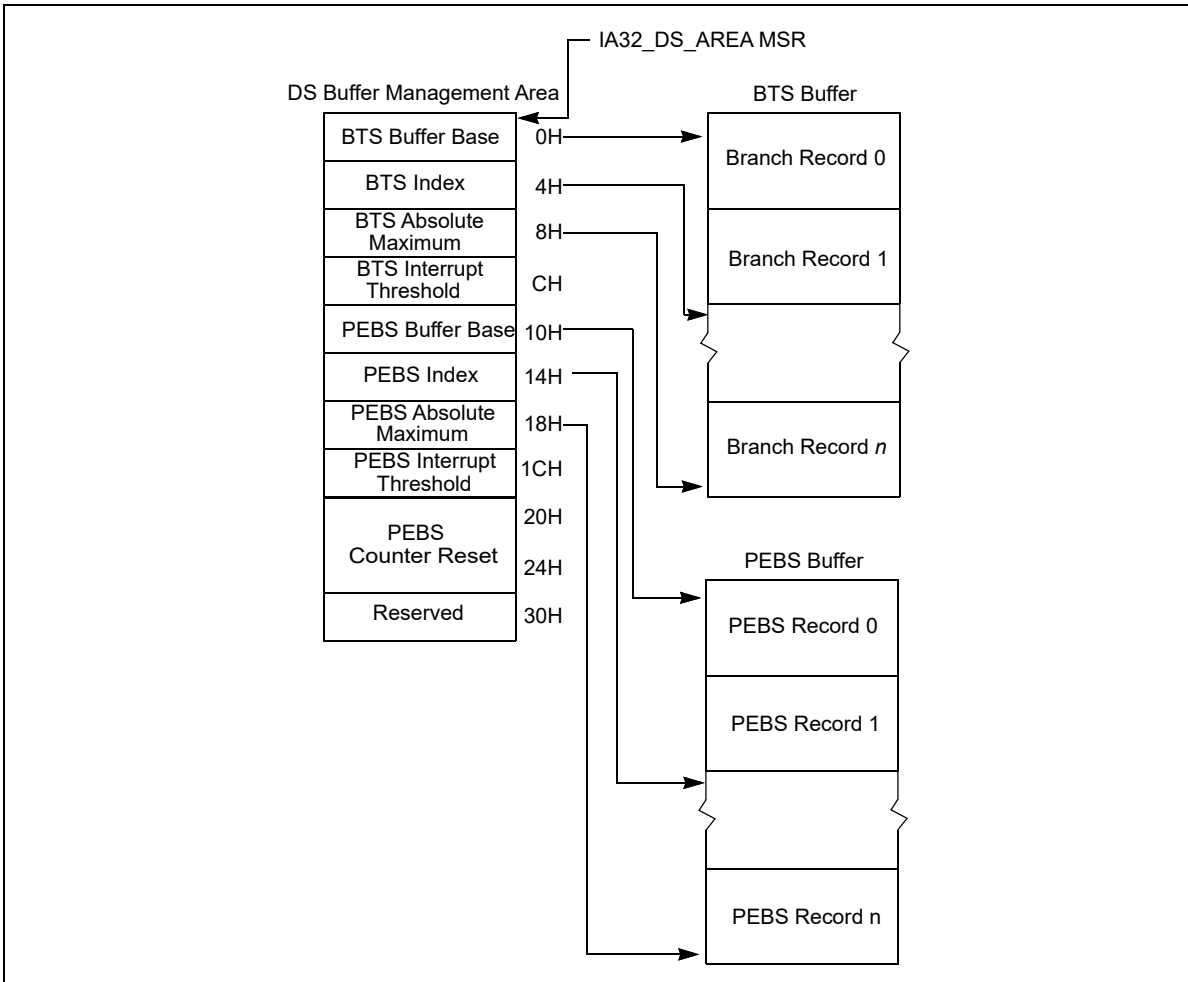
The DS save area and recording mechanism are disabled on INIT, processor Reset or transition to system-management mode (SMM) or IA-32e mode. It is similarly disabled on the generation of a machine-check exception on 45nm and 32nm Intel Atom processors and on processors with Netburst or Intel Core microarchitecture.

The BTS and PEBS facilities may not be available on all processors. The availability of these facilities is indicated by the `BTS_UNAVAILABLE` and `PEBS_UNAVAILABLE` flags, respectively, in the `IA32_MISC_ENABLE` MSR (see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4).

The DS save area is divided into three parts: buffer management area, branch trace store (BTS) buffer, and PEBS buffer (see Figure 20-5). The buffer management area is used to define the location and size of the BTS and PEBS buffers. The processor then uses the buffer management area to keep track of the branch and/or PEBS records in their respective buffers and to record the performance counter reset value. The linear address of the first byte of the DS buffer management area is specified with the `IA32_DS_AREA` MSR.

The fields in the buffer management area are as follows:

- **BTS buffer base** — Linear address of the first byte of the BTS buffer. This address should point to a natural doubleword boundary.
- **BTS index** — Linear address of the first byte of the next BTS record to be written to. Initially, this address should be the same as the address in the BTS buffer base field.
- **BTS absolute maximum** — Linear address of the next byte past the end of the BTS buffer. This address should be a multiple of the BTS record size (12 bytes) plus 1.
- **BTS interrupt threshold** — Linear address of the BTS record on which an interrupt is to be generated. This address must point to an offset from the BTS buffer base that is a multiple of the BTS record size. Also, it must be several records short of the BTS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the BTS absolute maximum record.
- **PEBS buffer base** — Linear address of the first byte of the PEBS buffer. This address should point to a natural doubleword boundary.
- **PEBS index** — Linear address of the first byte of the next PEBS record to be written to. Initially, this address should be the same as the address in the PEBS buffer base field.
- **PEBS absolute maximum** — Linear address of the next byte past the end of the PEBS buffer. This address should be a multiple of the PEBS record size (40 bytes) plus 1.
- **PEBS interrupt threshold** — Linear address of the PEBS record on which an interrupt is to be generated. This address must point to an offset from the PEBS buffer base that is a multiple of the PEBS record size. Also, it must be several records short of the PEBS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the PEBS absolute maximum record.
- **PEBS counter reset value** — A 64-bit value that the counter is to be set to when a PEBS record is written. Bits beyond the size of the counter are ignored. This value allows state information to be collected regularly every time the specified number of events occur.

Figure 20-5. DS Save Area Example¹**NOTES:**

1. This example represents the format for a system that supports PEBS on only one counter.

Figure 20-6 shows the structure of a 12-byte branch record in the BTS buffer. The fields in each record are as follows:

- **Last branch from** — Linear address of the instruction from which the branch, interrupt, or exception was taken.
- **Last branch to** — Linear address of the branch target or the first instruction in the interrupt or exception service routine.
- **Branch predicted** — Bit 4 of field indicates whether the branch that was taken was predicted (set) or not predicted (clear).

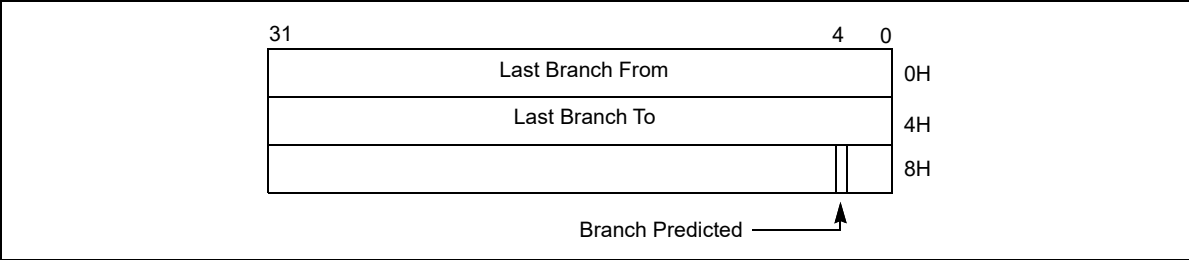


Figure 20-6. 32-bit Branch Trace Record Format

Figure 20-7 shows the structure of the 40-byte PEBS records. Nominally the register values are those at the beginning of the instruction that caused the event. However, there are cases where the registers may be logged in a partially modified state. The linear IP field shows the value in the EIP register translated from an offset into the current code segment to a linear address.

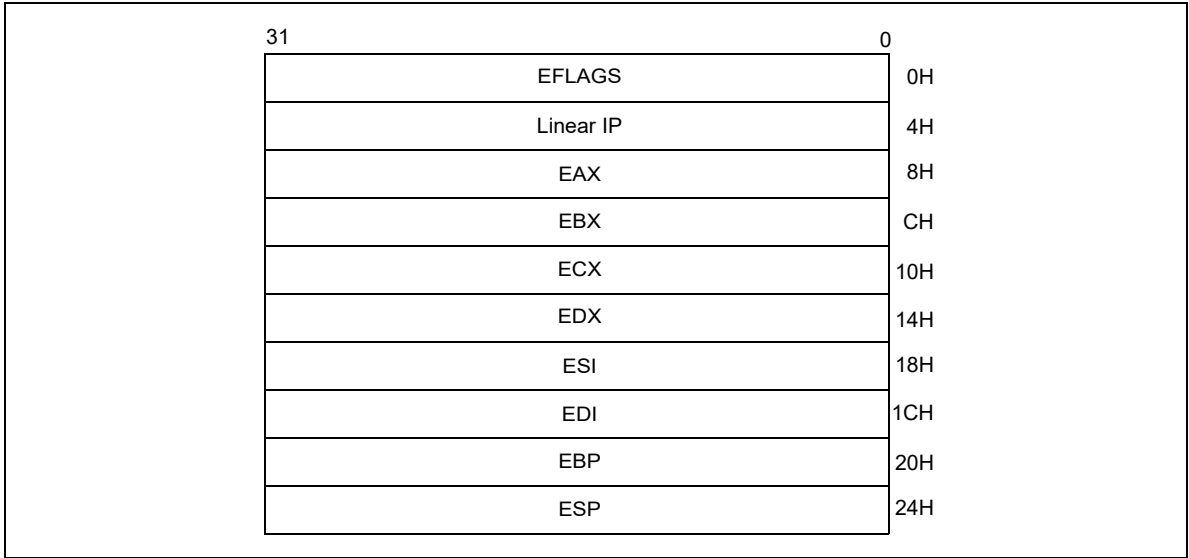


Figure 20-7. PEBS Record Format

20.4.9.1 64 Bit Format of the DS Save Area

When DTES64 = 1 (CPUID.01H:ECX[2] = 1), the structure of the DS save area is shown in Figure 20-8. When DTES64 = 0 (CPUID.01H:ECX[2] = 0) and IA-32e mode is active, the structure of the DS save area is shown in Figure 20-8. If IA-32e mode is not active the structure of the DS save area is as shown in Figure 20-5.

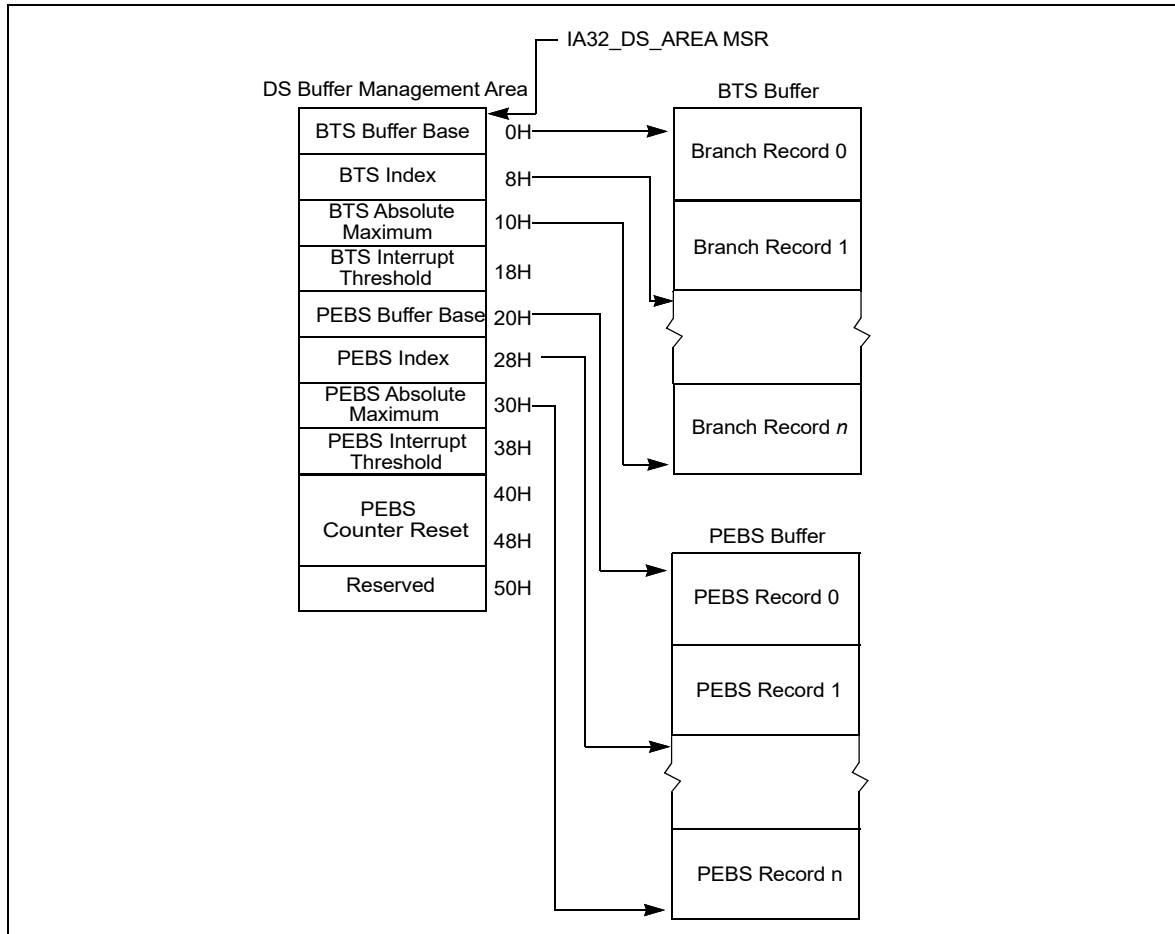


Figure 20-8. IA-32e Mode DS Save Area Example¹

NOTES:

1. This example represents the format for a system that supports PEBS on only one counter.

The IA32_DS_AREA MSR holds the 64-bit linear address of the first byte of the DS buffer management area. The structure of a branch trace record is similar to that shown in Figure 20-6, but each field is 8 bytes in length. This makes each BTS record 24 bytes (see Figure 20-9). The structure of a PEBS record is similar to that shown in Figure 20-7, but each field is 8 bytes in length and architectural states include register R8 through R15. This makes the size of a PEBS record in 64-bit mode 144 bytes (see Figure 20-10).

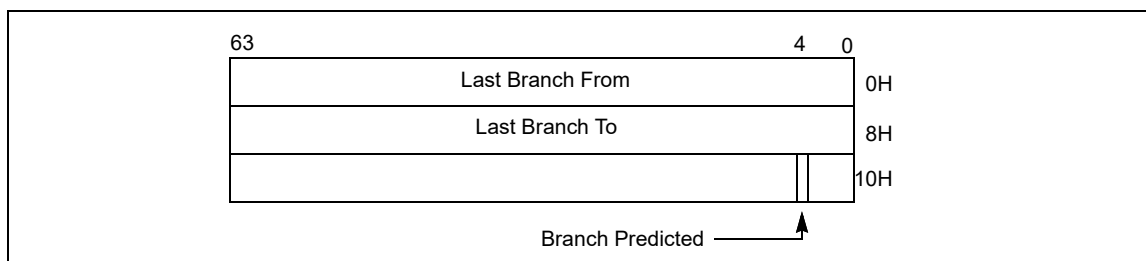


Figure 20-9. 64-bit Branch Trace Record Format

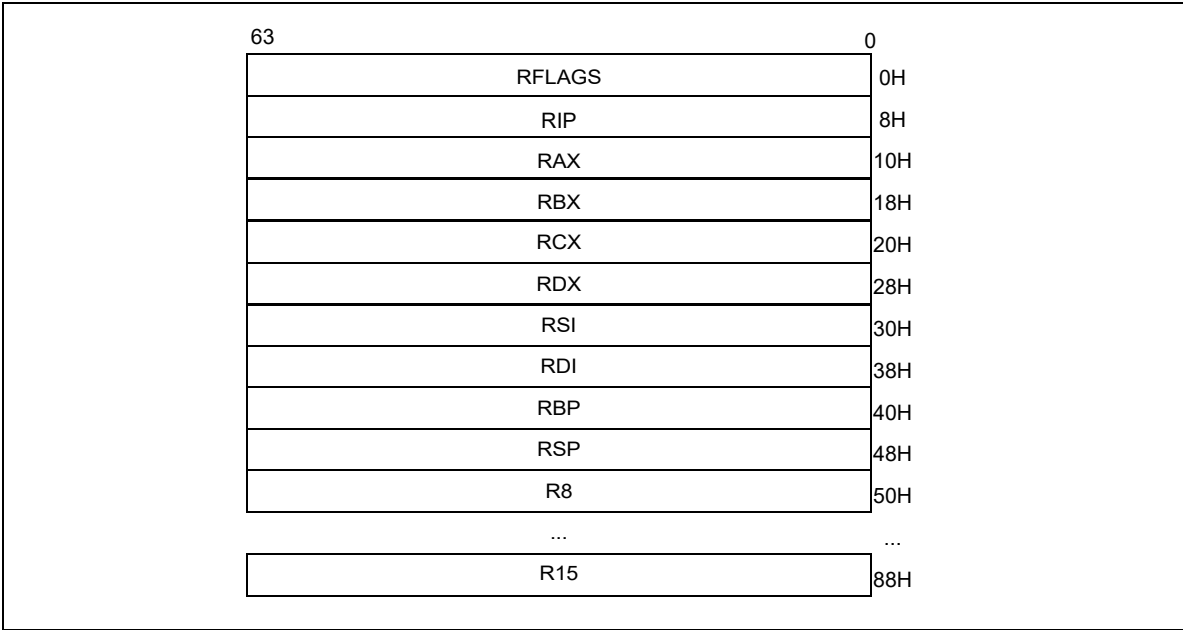


Figure 20-10. 64-bit PEBS Record Format

Fields in the buffer management area of a DS save area are described in Section 20.4.9.

The format of a branch trace record and a PEBS record are the same as the 64-bit record formats shown in Figures 20-9 and Figures 20-10, with the exception that the branch predicted bit is not supported by Intel Core microarchitecture or Intel Atom microarchitecture. The 64-bit record formats for BTS and PEBS apply to DS save area for all operating modes.

The procedures used to program IA32_DEBUGCTL MSR to set up a BTS buffer or a CPL-qualified BTS are described in Section 20.4.9.3 and Section 20.4.9.4.

Required elements for writing a DS interrupt service routine are largely the same on processors that support using DS Save area for BTS or PEBS records. However, on processors based on Intel NetBurst® microarchitecture, re-enabling counting requires writing to CCCRs. But a DS interrupt service routine on processors supporting architectural performance monitoring should:

- Re-enable the enable bits in IA32_PERF_GLOBAL_CTRL MSR if it is servicing an overflow PMI due to PEBS.
- Clear overflow indications by writing to IA32_PERF_GLOBAL_OVF_CTRL when a counting configuration is changed. This includes bit 62 (ClrOvfBuffer) and the overflow indication of counters used in either PEBS or general-purpose counting (specifically: bits 0 or 1; see Figures 22-3).

20.4.9.2 Setting Up the DS Save Area

To save branch records with the BTS buffer, the DS save area must first be set up in memory as described in the following procedure (See Section 22.6.2.4.1, “Setting up the PEBS Buffer,” for instructions for setting up a PEBS buffer, respectively, in the DS save area):

1. Create the DS buffer management information area in memory (see Section 20.4.9, “BTS and DS Save Area,” and Section 20.4.9.1, “64 Bit Format of the DS Save Area”). Also see the additional notes in this section.
2. Write the base linear address of the DS buffer management area into the IA32_DS_AREA MSR.
3. Set up the performance counter entry in the xAPIC LVT for fixed delivery and edge sensitive. See Section 13.5.1, “Local Vector Table.”
4. Establish an interrupt handler in the IDT for the vector associated with the performance counter entry in the xAPIC LVT.

5. Write an interrupt service routine to handle the interrupt. See Section 20.4.9.5, “Writing the DS Interrupt Service Routine.”

The following restrictions should be applied to the DS save area.

- The recording of branch records in the BTS buffer (or PEBS records in the PEBS buffer) may not operate properly if accesses to the linear addresses in any of the three DS save area sections cause page faults, VM exits, or the setting of accessed or dirty flags in the paging structures (ordinary or EPT). For that reason, system software should establish paging structures (both ordinary and EPT) to prevent such occurrences. Implications of this may be that an operating system should allocate this memory from a non-paged pool and that system software cannot do “lazy” page-table entry propagation for these pages. Some newer processor generations support “lazy” EPT page-table entry propagation for PEBS; see Section 22.3.9.1 and Section 22.9.5 for more information. A virtual-machine monitor may choose to allow use of PEBS by guest software only if EPT maps all guest-physical memory as present and read/write.
- The DS save area can be larger than a page, but the pages must be mapped to contiguous linear addresses. The buffer may share a page, so it need not be aligned on a 4-KByte boundary. For performance reasons, the base of the buffer must be aligned on a doubleword boundary and should be aligned on a cache line boundary.
- It is recommended that the buffer size for the BTS buffer and the PEBS buffer be an integer multiple of the corresponding record sizes.
- The precise event records buffer should be large enough to hold the number of precise event records that can occur while waiting for the interrupt to be serviced.
- The DS save area should be in kernel space. It must not be on the same page as code, to avoid triggering self-modifying code actions.
- There are no memory type restrictions on the buffers, although it is recommended that the buffers be designated as WB memory type for performance considerations.
- Either the system must be prevented from entering A20M mode while DS save area is active, or bit 20 of all addresses within buffer bounds must be 0.
- Pages that contain buffers must be mapped to the same physical addresses for all processes, such that any change to control register CR3 will not change the DS addresses.
- The DS save area is expected to be used only on systems with an enabled APIC. The LVT Performance Counter entry in the APIC must be initialized to use an interrupt gate instead of the trap gate.

20.4.9.3 Setting Up the BTS Buffer

Three flags in the MSR_DEBUGCTLA MSR (see Table 20-5), IA32_DEBUGCTL (see Figure 20-3), or MSR_DEBUGCTLB (see Figure 20-16) control the generation of branch records and storing of them in the BTS buffer; these are TR, BTS, and BTINT. The TR flag enables the generation of BTMs. The BTS flag determines whether the BTMs are sent out on the system bus (clear) or stored in the BTS buffer (set). BTMs cannot be simultaneously sent to the system bus and logged in the BTS buffer. The BTINT flag enables the generation of an interrupt when the BTS buffer is full. When this flag is clear, the BTS buffer is a circular buffer.

Table 20-5. IA32_DEBUGCTL Flag Encodings

TR	BTS	BTINT	Description
0	X	X	Branch trace messages (BTMs) off
1	0	X	Generate BTMs
1	1	0	Store BTMs in the BTS buffer, used here as a circular buffer
1	1	1	Store BTMs in the BTS buffer, and generate an interrupt when the buffer is nearly full

The following procedure describes how to set up a DS Save area to collect branch records in the BTS buffer:

1. Place values in the BTS buffer base, BTS index, BTS absolute maximum, and BTS interrupt threshold fields of the DS buffer management area to set up the BTS buffer in memory.
2. Set the TR and BTS flags in the IA32_DEBUGCTL for Intel Core Solo and Intel Core Duo processors or later processors (or MSR_DEBUGCTLA MSR for processors based on Intel NetBurst Microarchitecture; or MSR_DEBUGCTLB for Pentium M processors).

3. Clear the BTINT flag in the corresponding IA32_DEBUGCTL (or MSR_DEBUGCTLA MSR; or MSR_DEBUGCTLB) if a circular BTS buffer is desired.

NOTES

If the buffer size is set to less than the minimum allowable value (i.e., BTS absolute maximum < 1 + size of BTS record), the results of BTS is undefined.

In order to prevent generating an interrupt, when working with circular BTS buffer, SW need to set BTS interrupt threshold to a value greater than BTS absolute maximum (fields of the DS buffer management area). It's not enough to clear the BTINT flag itself only.

20.4.9.4 Setting Up CPL-Qualified BTS

If the processor supports CPL-qualified last branch recording mechanism, the generation of branch records and storing of them in the BTS buffer are determined by: TR, BTS, BTS_OFF_OS, BTS_OFF_USR, and BTINT. The encoding of these five bits are shown in Table 20-6.

Table 20-6. CPL-Qualified Branch Trace Store Encodings

TR	BTS	BTS_OFF_OS	BTS_OFF_USR	BTINT	Description
0	X	X	X	X	Branch trace messages (BTMs) off
1	0	X	X	X	Generates BTMs but do not store BTMs
1	1	0	0	0	Store all BTMs in the BTS buffer, used here as a circular buffer
1	1	1	0	0	Store BTMs with CPL > 0 in the BTS buffer
1	1	0	1	0	Store BTMs with CPL = 0 in the BTS buffer
1	1	1	1	X	Generate BTMs but do not store BTMs
1	1	0	0	1	Store all BTMs in the BTS buffer; generate an interrupt when the buffer is nearly full
1	1	1	0	1	Store BTMs with CPL > 0 in the BTS buffer; generate an interrupt when the buffer is nearly full
1	1	0	1	1	Store BTMs with CPL = 0 in the BTS buffer; generate an interrupt when the buffer is nearly full

20.4.9.5 Writing the DS Interrupt Service Routine

The BTS, non-precise event-based sampling, and PEBS facilities share the same interrupt vector and interrupt service routine (called the debug store interrupt service routine or DS ISR). To handle BTS, non-precise event-based sampling, and PEBS interrupts: separate handler routines must be included in the DS ISR. Use the following guidelines when writing a DS ISR to handle BTS, non-precise event-based sampling, and/or PEBS interrupts.

- The DS interrupt service routine (ISR) must be part of a kernel driver and operate at a current privilege level of 0 to secure the buffer storage area.
- Because the BTS, non-precise event-based sampling, and PEBS facilities share the same interrupt vector, the DS ISR must check for all the possible causes of interrupts from these facilities and pass control on to the appropriate handler.

BTS and PEBS buffer overflow would be the sources of the interrupt if the buffer index matches/exceeds the interrupt threshold specified. Detection of non-precise event-based sampling as the source of the interrupt is accomplished by checking for counter overflow.

- There must be separate save areas, buffers, and state for each processor in an MP system.
- Upon entering the ISR, branch trace messages and PEBS should be disabled to prevent race conditions during access to the DS save area. This is done by clearing TR flag in the IA32_DEBUGCTL (or MSR_DEBUGCTLA MSR) and by clearing the precise event enable flag in the MSR_PEBS_ENABLE MSR. These settings should be restored to their original values when exiting the ISR.

- The processor will not disable the DS save area when the buffer is full and the circular mode has not been selected. The current DS setting must be retained and restored by the ISR on exit.
- After reading the data in the appropriate buffer, up to but not including the current index into the buffer, the ISR must reset the buffer index to the beginning of the buffer. Otherwise, everything up to the index will look like new entries upon the next invocation of the ISR.
- The ISR must clear the mask bit in the performance counter LVT entry.
- The ISR must re-enable the counters to count via IA32_PERF_GLOBAL_CTRL/IA32_PERF_GLOBAL_OVF_CTRL if it is servicing an overflow PMI due to PEBS (or via CCCR's ENABLE bit on processor based on Intel NetBurst microarchitecture).
- The Pentium 4 Processor and Intel Xeon Processor mask PMIs upon receiving an interrupt. Clear this condition before leaving the interrupt handler.

20.5 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (INTEL® CORE™ 2 DUO AND INTEL ATOM® PROCESSORS)

The Intel Core 2 Duo processor family and Intel Xeon processors based on Intel Core microarchitecture or enhanced Intel Core microarchitecture provide last branch interrupt and exception recording. The facilities described in this section also apply to 45 nm and 32 nm Intel Atom processors. These capabilities are similar to those found in Pentium 4 processors, including support for the following facilities:

- **Debug Trace and Branch Recording Control** — The IA32_DEBUGCTL MSR provide bit fields for software to configure mechanisms related to debug trace, branch recording, branch trace store, and performance counter operations. See Section 20.4.1 for a description of the flags. See Figure 20-3 for the MSR layout.
- **Last branch record (LBR) stack** — There are a collection of MSR pairs that store the source and destination addresses related to recently executed branches. See Section 20.5.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts**
 - See Section 20.4.2 and Section 20.4.3. In addition, the ability to freeze the LBR stack on a PMI request is available.
 - 45 nm and 32 nm Intel Atom processors clear the TR flag when the FREEZE_LBRS_ON_PMI flag is set.
- **Branch trace messages** — See Section 20.4.4.
- **Last exception records** — See Section 20.13.3.
- **Branch trace store and CPL-qualified BTS** — See Section 20.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — see Section 20.4.7 for legacy Freeze_LBRS_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — see Section 20.4.7 for legacy Freeze_PerfMon_On_PMI operation.
- **FREEZE_WHILE_SMM (bit 14)** — FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 20.4.1.

20.5.1 LBR Stack

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported across Intel Core 2, Intel Atom processor families, and Intel processors based on Intel NetBurst microarchitecture.

Four pairs of MSRs are supported in the LBR stack for Intel Core 2 processors families and Intel processors based on Intel NetBurst microarchitecture:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 40H) through MSR_LASTBRANCH_3_FROM_IP (address 43H) store source addresses
 - MSR_LASTBRANCH_0_TO_IP (address 60H) through MSR_LASTBRANCH_3_TO_IP (address 63H) store destination addresses

- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 2 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

Eight pairs of MSRs are supported in the LBR stack for 45 nm and 32 nm Intel Atom processors:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 40H) through MSR_LASTBRANCH_7_FROM_IP (address 47H) store source addresses
 - MSR_LASTBRANCH_0_TO_IP (address 60H) through MSR_LASTBRANCH_7_TO_IP (address 67H) store destination addresses
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 3 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

The address format written in the FROM_IP/TO_IP MSRs may differ between processors. Software should query IA32_PERF_CAPABILITIES[5:0] and consult Section 20.4.8.1. The behavior of the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs corresponds to that of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

20.5.2 LBR Stack in Intel Atom® Processors based on the Silvermont Microarchitecture

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported in Intel Atom processors based on the Silvermont and Airmont microarchitectures. Eight pairs of MSRs are supported in the LBR stack.

LBR filtering is supported. Filtering of LBRs based on a combination of CPL and branch type conditions is supported. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT. The layout of MSR_LBR_SELECT is described in Table 20-11.

20.6 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON GOLDMONT MICROARCHITECTURE

Processors based on the Goldmont microarchitecture extend the capabilities described in Section 20.5.2 with the following enhancements:

- Supports new LBR format encoding 00110b in IA32_PERF_CAPABILITIES[5:0].
- Size of LBR stack increased to 32. Each entry includes MSR_LASTBRANCH_x_FROM_IP (address 0x680..0x69f) and MSR_LASTBRANCH_x_TO_IP (address 0x6c0..0x6df).
- LBR call stack filtering supported. The layout of MSR_LBR_SELECT is described in Table 20-13.
- Elapsed cycle information is added to MSR_LASTBRANCH_x_TO_IP. Format is shown in Table 20-7.
- Misprediction info is reported in the upper bits of MSR_LASTBRANCH_x_FROM_IP. MISPREDE bit format is shown in Table 20-8.
- Streamlined Freeze_LBRs_On_PMI operation; see Section 20.12.2.
- LBR MSRs may be cleared when MWAIT is used to request a C-state that is numerically higher than C1; see Section 20.12.3.

Table 20-7. MSR_LASTBRANCH_x_TO_IP for the Goldmont Microarchitecture

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch to” address. See Section 20.4.8.1 for address format.
Cycle Count (Saturating)	63:48	R/W	Elapsed core clocks since last update to the LBR stack.

20.7 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON GOLDMONT PLUS MICROARCHITECTURE

Next generation Intel Atom processors are based on the Goldmont Plus microarchitecture. Processors based on the Goldmont Plus microarchitecture extend the capabilities described in Section 20.6 with the following changes:

- Enumeration of new LBR format: encoding 00111b in IA32_PERF_CAPABILITIES[5:0] is supported, see Section 20.4.8.1.
- Each LBR stack entry consists of three MSRs:
 - MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 20-9.
 - MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 20-9.
 - MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data. Layout is the same as Table 20-16.

20.8 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR INTEL® XEON PHI™ PROCESSOR 7200/5200/3200

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported in the Intel® Xeon Phi™ processor 7200/5200/3200 series based on the Knights Landing microarchitecture. Eight pairs of MSRs are supported in the LBR stack, per thread:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 680H) through MSR_LASTBRANCH_7_FROM_IP (address 687H) store source addresses.
 - MSR_LASTBRANCH_0_TO_IP (address 6C0H) through MSR_LASTBRANCH_7_TO_IP (address 6C7H) store destination addresses.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 3 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

LBR filtering is supported. Filtering of LBRs based on a combination of CPL and branch type conditions is supported. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT. The layout of MSR_LBR_SELECT is described in Table 20-11.

The address format written in the FROM_IP/TO_IP MSRS may differ between processors. Software should query IA32_PERF_CAPABILITIES[5:0] and consult Section 20.4.8.1. The behavior of the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs corresponds to that of the LastExceptionToIP and LastExceptionFromIP MSRs found in the P6 family processors.

20.9 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON NEHALEM MICROARCHITECTURE

The processors based on Nehalem microarchitecture and Westmere microarchitecture support last branch interrupt and exception recording. These capabilities are similar to those found in Intel Core 2 processors and add additional capabilities:

- **Debug Trace and Branch Recording Control** — The IA32_DEBUGCTL MSR provides bit fields for software to configure mechanisms related to debug trace, branch recording, branch trace store, and performance counter operations. See Section 20.4.1 for a description of the flags. See Figure 20-11 for the MSR layout.
- **Last branch record (LBR) stack** — There are 16 MSR pairs that store the source and destination addresses related to recently executed branches. See Section 20.9.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts** — See Section 20.4.2 and Section 20.4.3. In addition, the ability to freeze the LBR stack on a PMI request is available.

- **Branch trace messages** — The IA32_DEBUGCTL MSR provides bit fields for software to enable each logical processor to generate branch trace messages. See Section 20.4.4. However, not all BTM messages are observable using the Intel® QPI link.
- **Last exception records** — See Section 20.13.3.
- **Branch trace store and CPL-qualified BTS** — See Section 20.4.6 and Section 20.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — see Section 20.4.7 for legacy Freeze_LBRs_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — see Section 20.4.7 for legacy Freeze_PerfMon_On_PMI operation.
- **UNCORE_PMI_EN (bit 13)** — When set, this logical processor is enabled to receive an counter overflow interrupt form the uncore.
- **FREEZE_WHILE_SMM (bit 14)** — FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 20.4.1.

Processors based on Nehalem microarchitecture provide additional capabilities:

- **Independent control of uncore PMI** — The IA32_DEBUGCTL MSR provides a bit field (see Figure 20-11) for software to enable each logical processor to receive an uncore counter overflow interrupt.
- **LBR filtering** — Processors based on Nehalem microarchitecture support filtering of LBR based on combination of CPL and branch type conditions. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT.

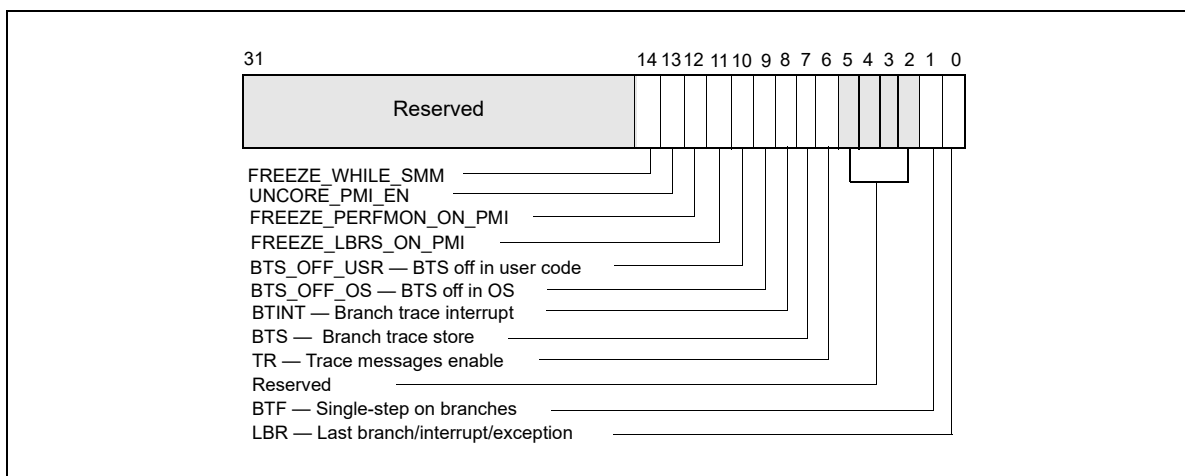


Figure 20-11. IA32_DEBUGCTL MSR for Processors Based on Nehalem Microarchitecture

20.9.1 LBR Stack

Processors based on Nehalem microarchitecture provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is shown in Table 20-8 and Table 20-9.

Table 20-8. MSR_LASTBRANCH_x_FROM_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the "branch from" address. See Section 20.4.8.1 for address format.
SIGN_EXT	62:48	R/W	Signed extension of bit 47 of this register.
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

Table 20-9. MSR_LASTBRANCH_x_TO_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch to” address. See Section 20.4.8.1 for address format
SIGN_EXT	63:48	R/W	Signed extension of bit 47 of this register.

Processors based on Nehalem microarchitecture have an LBR MSR Stack as shown in Table 20-10.

Table 20-10. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_1AH	16	0 to 15

20.9.2 Filtering of Last Branch Records

MSR_LBR_SELECT is cleared to zero at RESET, and LBR filtering is disabled, i.e., all branches will be captured. MSR_LBR_SELECT provides bit fields to specify the conditions of subsets of branches that will not be captured in the LBR. The layout of MSR_LBR_SELECT is shown in Table 20-11.

Table 20-11. MSR_LBR_SELECT for Nehalem Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps
FAR_BRANCH	8	R/W	When set, do not capture far branches
Reserved	63:9		Must be zero

20.10 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON SANDY BRIDGE MICROARCHITECTURE

Generally, all of the last branch record, interrupt, and exception recording facility described in Section 20.9, “Last Branch, Interrupt, and Exception Recording for Processors based on Nehalem Microarchitecture,” apply to processors based on Sandy Bridge microarchitecture. For processors based on Ivy Bridge microarchitecture, the same holds true.

One difference of note is that MSR_LBR_SELECT is shared between two logical processors in the same core. In Sandy Bridge microarchitecture, each logical processor has its own MSR_LBR_SELECT. The filtering semantics for “Near_ind_jmp” and “Near_rel_jmp” has been enhanced, see Table 20-12.

Table 20-12. MSR_LBR_SELECT for Sandy Bridge Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps except near indirect calls and near returns
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps except near relative calls.
FAR_BRANCH	8	R/W	When set, do not capture far branches
Reserved	63:9		Must be zero

20.11 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON HASWELL MICROARCHITECTURE

Generally, all of the last branch record, interrupt, and exception recording facility described in Section 20.10, “Last Branch, Interrupt, and Exception Recording for Processors based on Sandy Bridge Microarchitecture,” apply to next generation processors based on Haswell microarchitecture.

The LBR facility also supports an alternate capability to profile call stack profiles. Configuring the LBR facility to conduct call stack profiling is by writing 1 to the MSR_LBR_SELECT.EN_CALLSTACK[bit 9]; see Table 20-13. If MSR_LBR_SELECT.EN_CALLSTACK is clear, the LBR facility will capture branches normally as described in Section 20.10.

Table 20-13. MSR_LBR_SELECT for Haswell Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps except near indirect calls and near returns
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps except near relative calls.
FAR_BRANCH	8	R/W	When set, do not capture far branches
EN_CALLSTACK ¹	9		Enable LBR stack to use LIFO filtering to capture Call stack profile
Reserved	63:10		Must be zero

NOTES:

1. Must set valid combination of bits 0-8 in conjunction with bit 9 (as described below), otherwise the contents of the LBR MSRs are undefined.

The call stack profiling capability is an enhancement of the LBR facility. The LBR stack is a ring buffer typically used to profile control flow transitions resulting from branches. However, the finite depth of the LBR stack often become less effective when profiling certain high-level languages (e.g., C++), where a transition of the execution flow is accompanied by a large number of leaf function calls, each of which returns an individual parameter to form the list

of parameters for the main execution function call. A long list of such parameters returned by the leaf functions would serve to flush the data captured in the LBR stack, often losing the main execution context.

When the call stack feature is enabled, the LBR stack will capture unfiltered call data normally, but as return instructions are executed the last captured branch record is flushed from the on-chip registers in a last-in first-out (LIFO) manner. Thus, branch information relative to leaf functions will not be captured, while preserving the call stack information of the main line execution path.

The configuration of the call stack facility is summarized below:

- Set IA32_DEBUGCTL.LBR (bit 0) to enable the LBR stack to capture branch records. The source and target addresses of the call branches will be captured in the 16 pairs of From/To LBR MSRs that form the LBR stack.
- Program the Top of Stack (TOS) MSR that points to the last valid from/to pair. This register is incremented by 1, modulo 16, before recording the next pair of addresses.
- Program the branch filtering bits of MSR_LBR_SELECT (bits 0:8) as desired.
- Program the MSR_LBR_SELECT to enable LIFO filtering of return instructions with:
 - The following bits in MSR_LBR_SELECT must be set to '1': JCC, NEAR_IND_JMP, NEAR_REL_JMP, FAR_BRANCH, EN_CALLSTACK;
 - The following bits in MSR_LBR_SELECT must be cleared: NEAR_REL_CALL, NEAR-IND_CALL, NEAR_RET;
 - At most one of CPL_EQ_0, CPL_NEQ_0 is set.

Note that when call stack profiling is enabled, “zero length calls” are excluded from writing into the LBRs. (A “zero length call” uses the attribute of the call instruction to push the immediate instruction pointer on to the stack and then pops off that address into a register. This is accomplished without any matching return on the call.)

20.11.1 LBR Stack Enhancement

Processors based on Haswell microarchitecture provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is enumerated by IA32_PERF_CAPABILITIES[5:0] = 04H, and is shown in Table 20-14 and Table 20-9.

Table 20-14. MSR_LASTBRANCH_x_FROM_IP with TSX Information

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch from” address. See Section 20.4.8.1 for address format.
SIGN_EXT	60:48	R/W	Signed extension of bit 47 of this register.
TSX_ABORT	61	R/W	When set, indicates a TSX Abort entry LBR_FROM: EIP at the time of the TSX Abort LBR_TO: EIP of the start of HLE region, or EIP of the RTM Abort Handler
IN_TSX	62	R/W	When set, indicates the entry occurred in a TSX region
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

20.12 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON SKYLAKE MICROARCHITECTURE

Processors based on the Skylake microarchitecture provide a number of enhancement with storing last branch records:

- enumeration of new LBR format: encoding 00101b in IA32_PERF_CAPABILITIES[5:0] is supported, see Section 20.4.8.1.
- Each LBR stack entry consists of a triplets of MSRs:

- MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 20-9.
- MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 20-9.
- MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data.
- Size of LBR stack increased to 32.

Processors based on the Skylake microarchitecture support the same LBR filtering capabilities as described in Table 20-13.

Table 20-15. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_4EH, 06_5EH	32	0 to 31

20.12.1 MSR_LBR_INFO_x MSR

The layout of each MSR_LBR_INFO_x MSR is shown in Table 20-16.

Table 20-16. MSR_LBR_INFO_x

Bit Field	Bit Offset	Access	Description
Cycle Count (saturating)	15:0	R/W	Elapsed core clocks since last update to the LBR stack.
Reserved	60:16	R/W	Reserved
TSX_ABORT	61	R/W	When set, indicates a TSX Abort entry LBR_FROM: EIP at the time of the TSX Abort LBR_TO: EIP of the start of HLE region OR EIP of the RTM Abort Handler
IN_TSX	62	R/W	When set, indicates the entry occurred in a TSX region.
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

20.12.2 Streamlined Freeze_LBRs_On_PMI Operation

The FREEZE_LBRs_ON_PMI feature causes the LBRs to be frozen on a hardware request for a PMI. This prevents the LBRs from being overwritten by new branches, allowing the PMI handler to examine the control flow that preceded the PMI generation. Architectural performance monitoring version 4 and above supports a streamlined FREEZE_LBRs_ON_PMI operation for PMI service routine that replaces the legacy FREEZE_LBRs_ON_PMI operation (see Section 20.4.7).

While the legacy FREEZE_LBRs_ON_PMI clear the LBR bit in the IA32_DEBUGCTL MSR on a PMI request, the streamlined FREEZE_LBRs_ON_PMI will set the LBR_FRZ bit in IA32_PERF_GLOBAL_STATUS. Branches will not cause the LBRs to be updated when LBR_FRZ is set. Software can clear LBR_FRZ at the same time as it clears overflow bits by setting the LBR_FRZ bit as well as the needed overflow bit when writing to IA32_PERF_GLOBAL_STATUS_RESET MSR.

This streamlined behavior avoids race conditions between software and processor writes to IA32_DEBUGCTL that are possible with FREEZE_LBRs_ON_PMI clearing of the LBR enable.

20.12.3 LBR Behavior and Deep C-State

When MWAIT is used to request a C-state that is numerically higher than C1, then LBR state may be initialized to zero depending on optimized “waiting” state that is selected by the processor. The affected LBR states include the FROM, TO, INFO, LAST_BRANCH, LER, and LBR_TOS registers. The LBR enable bit and LBR_FROZEN bit are not affected. The LBR-time of the first LBR record inserted after an exit from such a C-state request will be zero.

20.13 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (PROCESSORS BASED ON INTEL NETBURST® MICROARCHITECTURE)

Pentium 4 and Intel Xeon processors based on Intel NetBurst microarchitecture provide the following methods for recording taken branches, interrupts, and exceptions:

- Store branch records in the last branch record (LBR) stack MSRs for the most recent taken branches, interrupts, and/or exceptions in MSRs. A branch record consists of a branch-from and a branch-to instruction address.
- Send the branch records out on the system bus as branch trace messages (BTMs).
- Log BTMs in a memory-resident branch trace store (BTS) buffer.

To support these functions, the processor provides the following MSRs and related facilities:

- **MSR_DEBUGCTLA MSR** — Enables last branch, interrupt, and exception recording; single-stepping on taken branches; branch trace messages (BTMs); and branch trace store (BTS). This register is named DebugCtlMSR in the P6 family processors.
- **Debug store (DS) feature flag (CPUID.01H:EDX.DS[21])** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer.
- **CPL-qualified debug store (DS) feature flag (CPUID.01H:ECX.DS_CPL[4])** — Indicates that the processor provides a CPL-qualified debug store (DS) mechanism, which allows software to selectively skip sending and storing BTMs, according to specified current privilege level settings, into a memory-resident BTS buffer.
- **IA32_MISC_ENABLE MSR** — Indicates that the processor provides the BTS facilities.
- **Last branch record (LBR) stack** — The LBR stack is a circular stack that consists of four MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. The LBR stack consists of 16 MSR pairs (MSR_LASTBRANCH_0_FROM_IP through MSR_LASTBRANCH_15_FROM_IP and MSR_LASTBRANCH_0_TO_IP through MSR_LASTBRANCH_15_TO_IP) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H].
- **Last branch record top-of-stack (TOS) pointer** — The TOS Pointer MSR contains a 2-bit pointer (0-3) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. This pointer becomes a 4-bit pointer (0-15) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H]. See also: Table 20-17, Figure 20-12, and Section 20.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **Last exception record** — See Section 20.13.3, “Last Exception Records.”

20.13.1 MSR_DEBUGCTLA MSR

The MSR_DEBUGCTLA MSR enables and disables the various last branch recording mechanisms described in the previous section. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode. A protected-mode operating system procedure is required to provide user access to this register. Figure 20-12 shows the flags in the MSR_DEBUGCTLA MSR. The functions of these flags are as follows:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. Each branch, interrupt, or exception is recorded as a 64-bit branch record. The processor clears this flag whenever a debug exception is generated (for example,

- when an instruction or data breakpoint or a single-step trap occurs). See Section 20.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 20.4.3, “Single-Stepping on Branches.”
 - **TR (trace message enable) flag (bit 2)** — When set, branch trace messages are enabled. Thereafter, when the processor detects a taken branch, interrupt, or exception, it sends the branch record out on the system bus as a branch trace message (BTM). See Section 20.4.4, “Branch Trace Messages.”

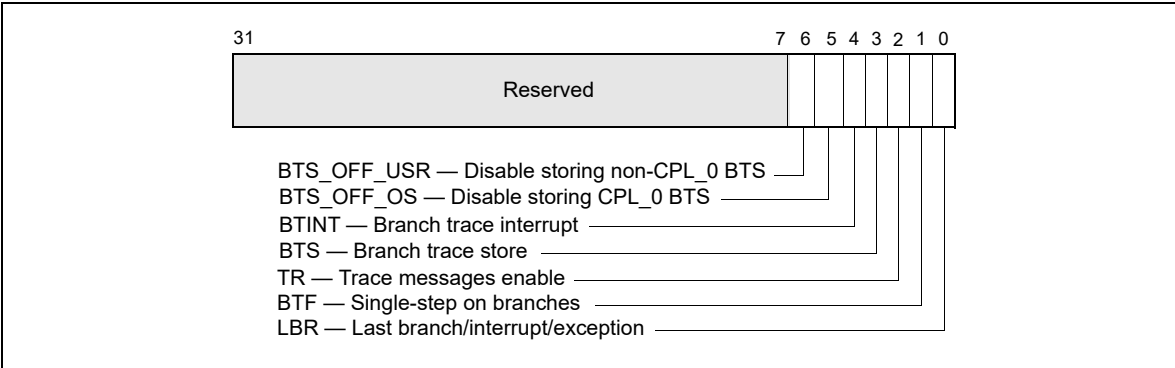


Figure 20-12. MSR_DEBUGCTLA MSR for Pentium 4 and Intel Xeon Processors

- **BTS (branch trace store) flag (bit 3)** — When set, enables the BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 20.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bits 4)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 20.4.5, “Branch Trace Store (BTS).”
- **BTS_OFF_OS (disable ring 0 branch trace store) flag (bit 5)** — When set, enables the BTS facilities to skip sending/logging CPL_0 BTMs to the memory-resident BTS buffer. See Section 20.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **BTS_OFF_USR (disable ring 0 branch trace store) flag (bit 6)** — When set, enables the BTS facilities to skip sending/logging non-CPL_0 BTMs to the memory-resident BTS buffer. See Section 20.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”

NOTE

The initial implementation of BTS_OFF_USR and BTS_OFF_OS in MSR_DEBUGCTLA is shown in Figure 20-12. The BTS_OFF_USR and BTS_OFF_OS fields may be implemented on other model-specific debug control register at different locations.

See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for a detailed description of each of the last branch recording MSRs.

20.13.2 LBR Stack for Processors Based on Intel NetBurst® Microarchitecture

The LBR stack is made up of LBR MSRs that are treated by the processor as a circular stack. The TOS pointer (MSR_LASTBRANCH_TOS MSR) points to the LBR MSR (or LBR MSR pair) that contains the most recent (last) branch record placed on the stack. Prior to placing a new branch record on the stack, the TOS is incremented by 1. When the TOS pointer reaches its maximum value, it wraps around to 0. See Table 20-17 and Figure 20-12.

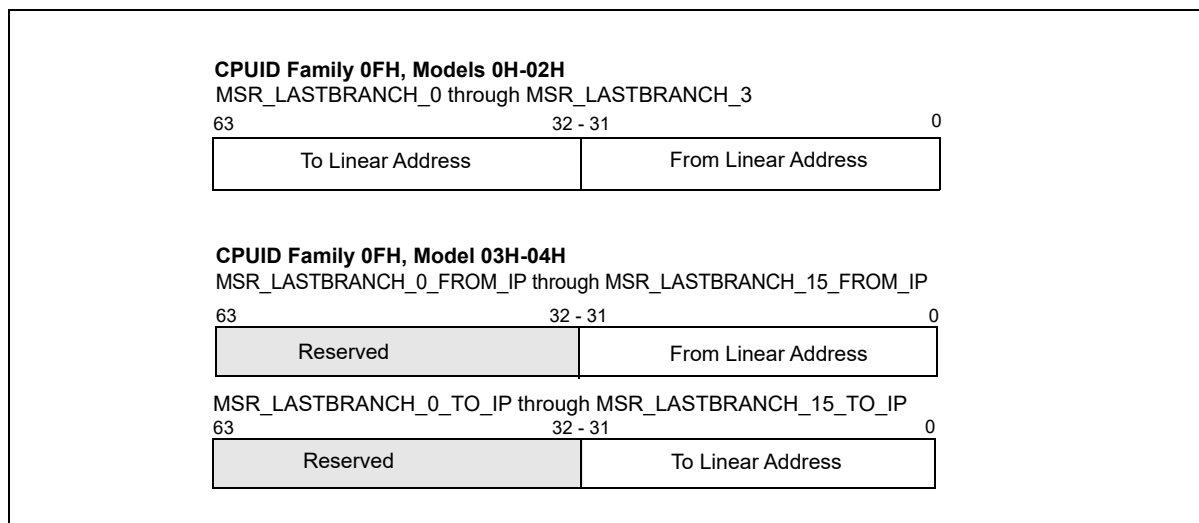
Table 20-17. LBR MSR Stack Size and TOS Pointer Range for the Pentium® 4 and the Intel® Xeon® Processor Family

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
Family 0FH, Models 0H-02H; MSRs at locations 1DBH-1DEH.	4	0 to 3
Family 0FH, Models; MSRs at locations 680H-68FH.	16	0 to 15
Family 0FH, Model 03H; MSRs at locations 6C0H-6CFH.	16	0 to 15

The registers in the LBR MSR stack and the MSR_LASTBRANCH_TOS MSR are read-only and can be read using the RDMSR instruction.

Figure 20-13 shows the layout of a branch record in an LBR MSR (or MSR pair). Each branch record consists of two linear addresses, which represent the “from” and “to” instruction pointers for a branch, interrupt, or exception. The contents of the from and to addresses differ, depending on the source of the branch:

- **Taken branch** — If the record is for a taken branch, the “from” address is the address of the branch instruction and the “to” address is the target instruction of the branch.
- **Interrupt** — If the record is for an interrupt, the “from” address the return instruction pointer (RIP) saved for the interrupt and the “to” address is the address of the first instruction in the interrupt handler routine. The RIP is the linear address of the next instruction to be executed upon returning from the interrupt handler.
- **Exception** — If the record is for an exception, the “from” address is the linear address of the instruction that caused the exception to be generated and the “to” address is the address of the first instruction in the exception handler routine.

**Figure 20-13. LBR MSR Branch Record Layout for the Pentium 4 and Intel® Xeon® Processor Family**

Additional information is saved if an exception or interrupt occurs in conjunction with a branch instruction. If a branch instruction generates a trap type exception, two branch records are stored in the LBR stack: a branch record for the branch instruction followed by a branch record for the exception.

If a branch instruction is immediately followed by an interrupt, a branch record is stored in the LBR stack for the branch instruction followed by a record for the interrupt.

20.13.3 Last Exception Records

The Pentium 4, Intel Xeon, Pentium M, Intel® Core™ Solo, Intel® Core™ Duo, Intel® Core™2 Duo, Intel® Core™ i7 and Intel Atom® processors provide two MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate the functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in the P6 family processors.

The MSR_LER_TO_LIP and MSR_LER_FROM_LIP MSRs contain a branch record for the last branch that the processor took prior to an exception or interrupt being generated.

20.14 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS)

Intel Core Solo and Intel Core Duo processors provide last branch interrupt and exception recording. This capability is almost identical to that found in Pentium 4 and Intel Xeon processors. There are differences in the stack and in some MSR names and locations.

Note the following:

- **IA32_DEBUGCTL MSR** — Enables debug trace interrupt, debug trace store, trace messages enable, performance monitoring breakpoint flags, single stepping on branches, and last branch. IA32_DEBUGCTL MSR is located at register address 01D9H.

See Figure 20-14 for the layout and the entries below for a description of the flags:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the “Last Branch Record (LBR) Stack” below.
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 20.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
- **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception; it sends the branch record out on the system bus as a branch trace message (BTM). See Section 20.4.4, “Branch Trace Messages,” for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7)** — When set, the flag enables BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 20.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bits 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 20.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

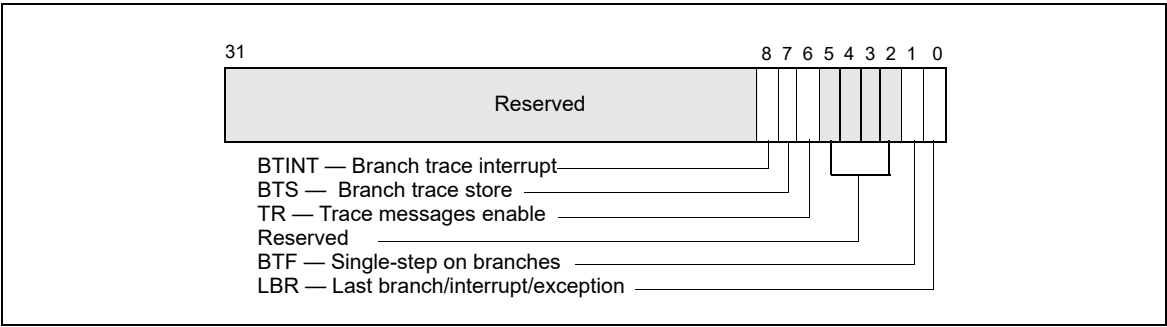


Figure 20-14. IA32_DEBUGCTL MSR for Intel® Core™ Solo and Intel® Core™ Duo Processors

- **Debug store (DS) feature flag (bit 21), returned by the CPUID instruction** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer. See Section 20.4.5, “Branch Trace Store (BTS).”
- **Last Branch Record (LBR) Stack** — The LBR stack consists of 8 MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_7); bits 31-0 hold the ‘from’ address, bits 63-32 hold the ‘to’ address (MSR addresses start at 40H). See Figure 20-15.

- **Last Branch Record Top-of-Stack (TOS) Pointer** — The TOS Pointer MSR contains a 3-bit pointer (bits 2-0) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. For Intel Core Solo and Intel Core Duo processors, this MSR is located at register address 01C9H.

For compatibility, the Intel Core Solo and Intel Core Duo processors provide two 32-bit MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

For details, see Section 20.12, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture,” and Section 2.20, “MSRs In Intel® Core™ Solo and Intel® Core™ Duo Processors,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

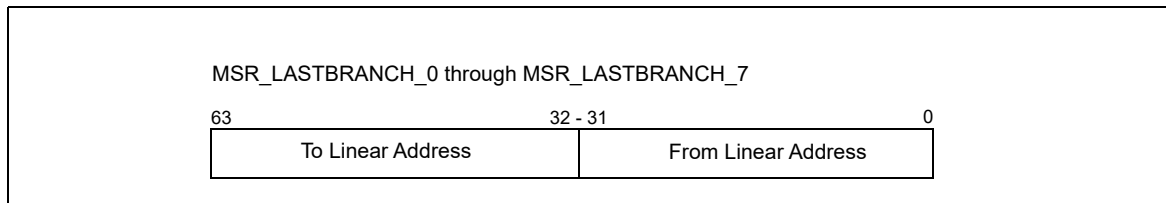


Figure 20-15. LBR Branch Record Layout for the Intel® Core™ Solo and Intel® Core™ Duo Processor

20.15 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (PENTIUM M PROCESSORS)

Like the Pentium 4 and Intel Xeon processor family, Pentium M processors provide last branch interrupt and exception recording. The capability operates almost identically to that found in Pentium 4 and Intel Xeon processors. There are differences in the shape of the stack and in some MSR names and locations. Note the following:

- **MSR_DEBUGCTLB MSR** — Enables debug trace interrupt, debug trace store, trace messages enable, performance monitoring breakpoint flags, single stepping on branches, and last branch. For Pentium M processors, this MSR is located at register address 01D9H. See Figure 20-16 and the entries below for a description of the flags.
 - **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the “Last Branch Record (LBR) Stack” bullet below.
 - **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 20.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
 - **PBi (performance monitoring/breakpoint pins) flags (bits 5-2)** — When these flags are set, the performance monitoring/breakpoint pins on the processor (BP0#, BP1#, BP2#, and BP3#) report breakpoint matches in the corresponding breakpoint-address registers (DR0 through DR3). The processor asserts then deasserts the corresponding BPi# pin when a breakpoint match occurs. When a PBi flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
 - **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception, it sends the branch record out on the system bus as a branch trace message (BTM). See Section 20.4.4, “Branch Trace Messages,” for more information about the TR flag.
 - **BTS (branch trace store) flag (bit 7)** — When set, enables the BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 20.4.9, “BTS and DS Save Area.”
 - **BTINT (branch trace interrupt) flag (bits 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 20.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

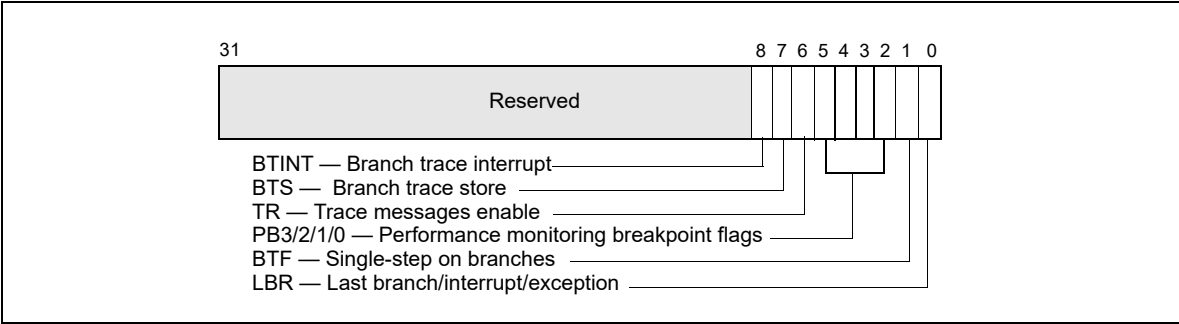


Figure 20-16. MSR_DEBUGCTLB MSR for Pentium M Processors

- **Debug store (DS) feature flag (bit 21), returned by the CPUID instruction** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer. See Section 20.4.5, “Branch Trace Store (BTS).”
- **Last Branch Record (LBR) Stack** — The LBR stack consists of 8 MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_7); bits 31-0 hold the ‘from’ address, bits 63-32 hold the ‘to’ address. For Pentium M Processors, these pairs are located at register addresses 040H-047H. See Figure 20-17.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The TOS Pointer MSR contains a 3-bit pointer (bits 2-0) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. For Pentium M Processors, this MSR is located at register address 01C9H.

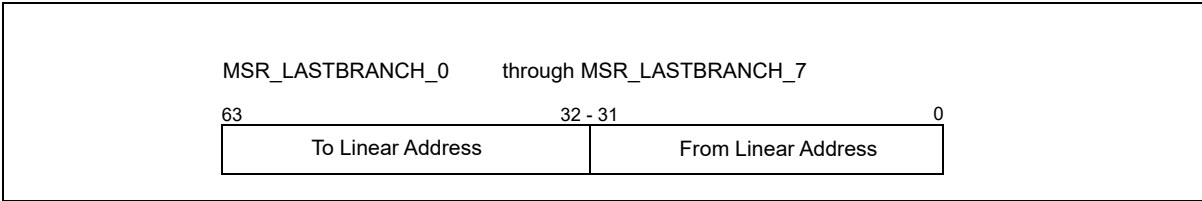


Figure 20-17. LBR Branch Record Layout for the Pentium M Processor

For more detail on these capabilities, see Section 20.13.3, “Last Exception Records,” and Section 2.21, “MSRs In the Pentium M Processor,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

20.16 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (P6 FAMILY PROCESSORS)

The P6 family processors provide five MSRs for recording the last branch, interrupt, or exception taken by the processor: DEBUGCTLMR, LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP. These registers can be used to collect last branch records, to set breakpoints on branches, interrupts, and exceptions, and to single-step from one branch to the next.

See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for a detailed description of each of the last branch recording MSRs.

20.16.1 DEBUGCTLMR Register

The version of the DEBUGCTLMR register found in the P6 family processors enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode.

A protected-mode operating system procedure is required to provide user access to this register. Figure 20-18 shows the flags in the DEBUGCTLMSR register for the P6 family processors. The functions of these flags are as follows:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records the source and target addresses (in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs) for the last branch and the last exception or interrupt taken by the processor prior to a debug exception being generated. The processor clears this flag whenever a debug exception, such as an instruction or data breakpoint or single-step trap occurs.

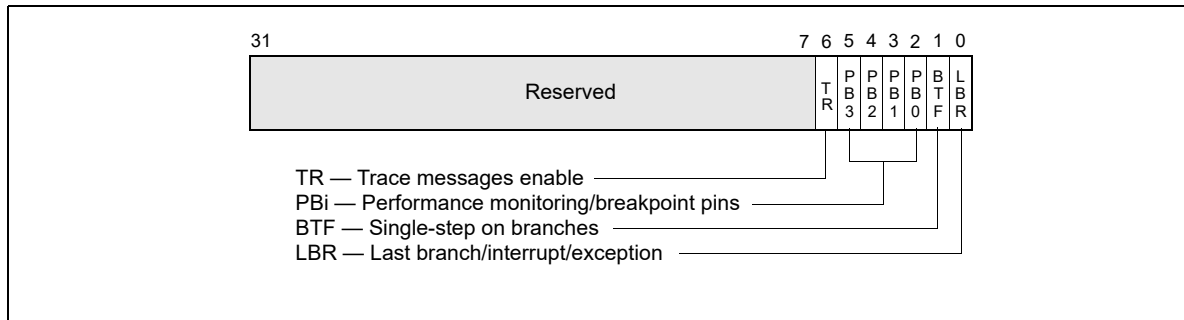


Figure 20-18. DEBUGCTLMSR Register (P6 Family Processors)

- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag. See Section 20.4.3, “Single-Stepping on Branches.”
- **P_Bi (performance monitoring/breakpoint pins) flags (bits 2 through 5)** — When these flags are set, the performance monitoring/breakpoint pins on the processor (BP0#, BP1#, BP2#, and BP3#) report breakpoint matches in the corresponding breakpoint-address registers (DR0 through DR3). The processor asserts then deasserts the corresponding P_Bi# pin when a breakpoint match occurs. When a P_Bi flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
- **TR (trace message enable) flag (bit 6)** — When set, trace messages are enabled as described in Section 20.4.4, “Branch Trace Messages.” Setting this flag greatly reduces the performance of the processor. When trace messages are enabled, the values stored in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are undefined.

20.16.2 Last Branch and Last Exception MSRs

The LastBranchToIP and LastBranchFromIP MSRs are 32-bit registers for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated. When a branch occurs, the processor loads the address of the branch instruction into the LastBranchFromIP MSR and loads the target address for the branch into the LastBranchToIP MSR.

When an interrupt or exception occurs (other than a debug exception), the address of the instruction that was interrupted by the exception or interrupt is loaded into the LastBranchFromIP MSR and the address of the exception or interrupt handler that is called is loaded into the LastBranchToIP MSR.

The LastExceptionToIP and LastExceptionFromIP MSRs (also 32-bit registers) record the instruction pointers for the last branch that the processor took prior to an exception or interrupt being generated. When an exception or interrupt occurs, the contents of the LastBranchToIP and LastBranchFromIP MSRs are copied into these registers before the to and from addresses of the exception or interrupt are recorded in the LastBranchToIP and LastBranchFromIP MSRs.

These registers can be read using the RDMSR instruction.

Note that the values stored in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are offsets into the current code segment, as opposed to linear addresses, which are saved in last branch records for the Pentium 4 and Intel Xeon processors.

20.16.3 Monitoring Branches, Exceptions, and Interrupts

When the LBR flag in the DEBUGCTLMSR register is set, the processor automatically begins recording branches that it takes, exceptions that are generated (except for debug exceptions), and interrupts that are serviced. Each time a branch, exception, or interrupt occurs, the processor records the to and from instruction pointers in the LastBranchToIP and LastBranchFromIP MSRs. In addition, for interrupts and exceptions, the processor copies the contents of the LastBranchToIP and LastBranchFromIP MSRs into the LastExceptionToIP and LastExceptionFromIP MSRs prior to recording the to and from addresses of the interrupt or exception.

When the processor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler, but does not touch the last branch and last exception MSRs. The addresses for the last branch, interrupt, or exception taken are thus retained in the LastBranchToIP and LastBranchFromIP MSRs and the addresses of the last branch prior to an interrupt or exception are retained in the LastExceptionToIP, and LastExceptionFromIP MSRs.

The debugger can use the last branch, interrupt, and/or exception addresses in combination with code-segment selectors retrieved from the stack to reset breakpoints in the breakpoint-address registers (DR0 through DR3), allowing a backward trace from the manifestation of a particular bug toward its source. Because the instruction pointers recorded in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are offsets into a code segment, software must determine the segment base address of the code segment associated with the control transfer to calculate the linear address to be placed in the breakpoint-address registers. The segment base address can be determined by reading the segment selector for the code segment from the stack and using it to locate the segment descriptor for the segment in the GDT or LDT. The segment base address can then be read from the segment descriptor.

Before resuming program execution from a debug-exception handler, the handler must set the LBR flag again to re-enable last branch and last exception/interrupt recording.

20.17 TIME-STAMP COUNTER

The Intel 64 and IA-32 architectures (beginning with the Pentium processor) define a time-stamp counter mechanism that can be used to monitor and identify the relative time occurrence of processor events. The counter's architecture includes the following components:

- **TSC flag** — A feature bit that indicates the availability of the time-stamp counter. The counter is available in an if the function CPUID.01H:EDX.TSC[4] = 1.
- **IA32_TIME_STAMP_COUNTER MSR** (called TSC MSR in P6 family and Pentium processors) — The MSR used as the counter.
- **RDTSC instruction** — An instruction used to read the time-stamp counter.
- **TSD flag** — A control register flag is used to enable or disable the time-stamp counter (enabled if CR4.TSD[bit 2] = 1).

The time-stamp counter (as implemented in the P6 family, Pentium, Pentium M, Pentium 4, Intel Xeon, Intel Core Solo and Intel Core Duo processors and later processors) is a 64-bit counter that is set to 0 following a RESET of the processor. Following a RESET, the counter increments even when the processor is halted by the HLT instruction or the external STPCLK# pin. Note that the assertion of the external DPSLP# pin may cause the time-stamp counter to stop.

Processor families increment the time-stamp counter differently:

- For Pentium M processors (family [06H], models [09H, 0DH]); for Pentium 4 processors, Intel Xeon processors (family [0FH], models [00H, 01H, or 02H]); and for P6 family processors: the time-stamp counter increments with every internal processor clock cycle.

The internal processor clock cycle is determined by the current core-clock to bus-clock ratio. Intel® SpeedStep® technology transitions may also impact the processor clock.

- For Pentium 4 processors, Intel Xeon processors (family [0FH], models [03H and higher]); for Intel Core Solo and Intel Core Duo processors (family [06H], model [0EH]); for the Intel Xeon processor 5100 series and Intel Core 2 Duo processors (family [06H], model [0FH]); for Intel Core 2 and Intel Xeon processors (family [06H], DisplayModel [17H]); for Intel Atom processors (family [06H], DisplayModel [1CH]): the time-stamp counter increments at a constant rate. That rate may be set by the maximum core-clock to bus-clock ratio of the

processor or may be set by the maximum resolved frequency at which the processor is booted. The maximum resolved frequency may differ from the processor base frequency, see Section 22.7.2 for more detail. On certain processors, the TSC frequency may not be the same as the frequency in the brand string.

The specific processor configuration determines the behavior. Constant TSC behavior ensures that the duration of each clock tick is uniform and supports the use of the TSC as a wall clock timer even if the processor core changes frequency. This is the architectural behavior moving forward.

NOTE

To determine average processor clock frequency, Intel recommends the use of performance monitoring logic to count processor core clocks over the period of time for which the average is required. See Section 22.6.4.5, “Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture,” and <https://perfmon-events.intel.com/> for more information.

The RDTSC instruction reads the time-stamp counter and is guaranteed to return a monotonically increasing unique value whenever executed, except for a 64-bit counter wraparound. Intel guarantees that the time-stamp counter will not wraparound within 10 years after being reset. The period for counter wrap is longer for Pentium 4, Intel Xeon, P6 family, and Pentium processors.

Normally, the RDTSC instruction can be executed by programs and procedures running at any privilege level and in virtual-8086 mode. The TSD flag allows use of this instruction to be restricted to programs and procedures running at privilege level 0. A secure operating system would set the TSD flag during system initialization to disable user access to the time-stamp counter. An operating system that disables user access to the time-stamp counter should emulate the instruction through a user-accessible programming interface.

The RDTSC instruction is not serializing or ordered with other instructions. It does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDTSC instruction operation is performed.

The RDMSR and WRMSR instructions read and write the time-stamp counter, treating the time-stamp counter as an ordinary MSR (address 10H). In the Pentium 4, Intel Xeon, and P6 family processors, all 64-bits of the time-stamp counter are read using RDMSR (just as with RDTSC). When WRMSR is used to write the time-stamp counter on processors before family [0FH], models [03H, 04H]: only the low-order 32-bits of the time-stamp counter can be written (the high-order 32 bits are cleared to 0). For family [0FH], models [03H, 04H, 06H]; for family [06H]], model [0EH, 0FH]; for family [06H]], DisplayModel [17H, 1AH, 1CH, 1DH]: all 64 bits are writable.

20.17.1 Invariant TSC

The time stamp counter in newer processors may support an enhancement, referred to as invariant TSC. Processor's support for invariant TSC is indicated by CPUID.80000007H:EDX[8].

The invariant TSC will run at a constant rate in all ACPI P-, C-, and T-states. This is the architectural behavior moving forward. On processors with invariant TSC support, the OS may use the TSC for wall clock timer services (instead of ACPI or HPET timers). TSC reads are much more efficient and do not incur the overhead associated with a ring transition or access to a platform resource.

20.17.2 IA32_TSC_AUX Register and RDTSCP Support

Processors based on Nehalem microarchitecture provide an auxiliary TSC register, IA32_TSC_AUX that is designed to be used in conjunction with IA32_TSC. IA32_TSC_AUX provides a 32-bit field that is initialized by privileged software with a signature value (for example, a logical processor ID).

The primary usage of IA32_TSC_AUX in conjunction with IA32_TSC is to allow software to read the 64-bit time stamp in IA32_TSC and signature value in IA32_TSC_AUX with the instruction RDTSCP in an atomic operation. RDTSCP returns the 64-bit time stamp in EDX:EAX and the 32-bit TSC_AUX signature value in ECX. The atomicity of RDTSCP ensures that no context switch can occur between the reads of the TSC and TSC_AUX values.

Support for RDTSCP is indicated by CPUID.80000001H:EDX[27]. As with RDTSC instruction, non-ring 0 access is controlled by CR4.TSD (Time Stamp Disable flag).

User mode software can use RDTSCP to detect if CPU migration has occurred between successive reads of the TSC. It can also be used to adjust for per-CPU differences in TSC values in a NUMA system.

20.17.3 Time-Stamp Counter Adjustment

Software can modify the value of the time-stamp counter (TSC) of a logical processor by using the WRMSR instruction to write to the IA32_TIME_STAMP_COUNTER MSR (address 10H). Because such a write applies only to that logical processor, software seeking to synchronize the TSC values of multiple logical processors must perform these writes on each logical processor. It may be difficult for software to do this in a way that ensures that all logical processors will have the same value for the TSC at a given point in time.

The synchronization of TSC adjustment can be simplified by using the 64-bit IA32_TSC_ADJUST MSR (address 3BH). Like the IA32_TIME_STAMP_COUNTER MSR, the IA32_TSC_ADJUST MSR is maintained separately for each logical processor. A logical processor maintains and uses the IA32_TSC_ADJUST MSR as follows:

- On RESET, the value of the IA32_TSC_ADJUST MSR is 0.
- If an execution of WRMSR to the IA32_TIME_STAMP_COUNTER MSR adds (or subtracts) value X from the TSC, the logical processor also adds (or subtracts) value X from the IA32_TSC_ADJUST MSR.
- If an execution of WRMSR to the IA32_TSC_ADJUST MSR adds (or subtracts) value X from that MSR, the logical processor also adds (or subtracts) value X from the TSC.

Unlike the TSC, the value of the IA32_TSC_ADJUST MSR changes only in response to WRMSR (either to the MSR itself, or to the IA32_TIME_STAMP_COUNTER MSR). Its value does not otherwise change as time elapses. Software seeking to adjust the TSC can do so by using WRMSR to write the same value to the IA32_TSC_ADJUST MSR on each logical processor.

Processor support for the IA32_TSC_ADJUST MSR is indicated by CPUID.07H.00H:EBX.TSC_ADJUST (bit 1).

20.17.4 Invariant Time-Keeping

The invariant TSC is based on the invariant timekeeping hardware (called Always Running Timer or ART), that runs at the core crystal clock frequency. The ratio defined by CPUID.15H expresses the frequency relationship between the ART hardware and TSC.

If CPUID.15H:EBX[31:0] != 0 and CPUID.80000007H:EDX.TSC_INVARIANT = 1, the following linearity relationship holds between TSC and the ART hardware:

$$\text{TSC\_Value} = (\text{ART\_Value} * \text{CPUID.15H:EBX}[31:0]) / \text{CPUID.15H:EAX}[31:0] + K$$

Where 'K' is an offset that can be adjusted by a privileged agent¹.

When ART hardware is reset, both invariant TSC and K are also reset.

20.18 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) MONITORING FEATURES

The Intel Resource Director Technology (Intel RDT) feature set provides a set of monitoring capabilities including Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring (MBM). The Intel® Xeon® processor E5 v3 family introduced resource monitoring capability in each logical processor to measure specific platform shared resource metrics, for example, L3 cache occupancy. The programming interface for these monitoring features is described in this section. Two features within the monitoring feature set provided are described - Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring.

Cache Monitoring Technology (CMT) allows an Operating System, Hypervisor or similar system management agent to determine the usage of cache by applications running on the platform. The initial implementation is directed at L3 cache monitoring (currently the last level cache in most server platforms).

1. IA32_TSC_ADJUST MSR and the TSC-offset field in the VM execution controls of VMCS are some of the common interfaces that privileged software can use to manage the time stamp counter for keeping time

Memory Bandwidth Monitoring (MBM), introduced in the Intel® Xeon® processor E5 v4 family, builds on the CMT infrastructure to allow monitoring of bandwidth from one level of the cache hierarchy to the next - in this case focusing on the L3 cache, which is typically backed directly by system memory. As a result of this implementation, memory bandwidth can be monitored.

The monitoring mechanisms described provide the following key shared infrastructure features:

- A mechanism to enumerate the presence of the monitoring capabilities within the platform (via a CPUID feature bit).
- A framework to enumerate the details of each sub-feature (including CMT and MBM, as discussed later, via CPUID leaves and sub-leaves).
- A mechanism for the OS or Hypervisor to indicate a software-defined ID for each of the software threads (applications, virtual machines, etc.) that are scheduled to run on a logical processor. These identifiers are known as Resource Monitoring IDs (RMIDs).
- Mechanisms in hardware to monitor cache occupancy and bandwidth statistics as applicable to a given product generation on a per software-id basis.
- Mechanisms for the OS or Hypervisor to read back the collected metrics such as L3 occupancy or Memory Bandwidth for a given software ID at any point during runtime.

20.18.1 Overview of Cache Monitoring Technology and Memory Bandwidth Monitoring

The shared resource monitoring features described in this chapter provide a layer of abstraction between applications and logical processors through the use of **Resource Monitoring IDs** (RMIDs). Each logical processor in the system can be assigned an RMID independently, or multiple logical processors can be assigned to the same RMID value (e.g., to track an application with multiple threads). For each logical processor, only one RMID value is active at a time. This is enforced by the IA32_PQR_ASSOC MSR, which specifies the active RMID of a logical processor. Writing to this MSR by software changes the active RMID of the logical processor from an old value to a new value.

The underlying platform shared resource monitoring hardware tracks cache metrics such as cache utilization and misses as a result of memory accesses according to the RMIDs and reports monitored data via a counter register (IA32_QM_CTR). The specific event types supported vary by generation and can be enumerated via CPUID. To read back monitored data, software configures an event selection MSR (IA32_QM_EVTSEL) to specify which metric is to be reported and the specific RMID for which the data should be returned.

Processor support of the monitoring framework and sub-features such as CMT is reported via the CPUID instruction. The resource type available to the monitoring framework is enumerated via a new leaf function in CPUID. Reading and writing to the monitoring MSRs requires the RDMSR and WRMSR instructions.

The Cache Monitoring Technology feature set provides the following unique mechanisms:

- A mechanism to enumerate the presence and details of the CMT feature as applicable to a given level of the cache hierarchy, independent of other monitoring features.
- CMT-specific event codes to read occupancy for a given level of the cache hierarchy.

The Memory Bandwidth Monitoring feature provides the following unique mechanisms:

- A mechanism to enumerate the presence and details of the MBM feature as applicable to a given level of the cache hierarchy, independent of other monitoring features.
- MBM-specific event codes to read bandwidth out to the next level of the hierarchy and various sub-event codes to read more specific metrics as discussed later (e.g., total bandwidth vs. bandwidth only from local memory controllers on the same package).

20.18.2 Enabling Monitoring: Usage Flow

Figure 20-19 illustrates the key steps for OS/VMM to detect support of shared resource monitoring features such as CMT and enable resource monitoring for available resource types and monitoring events.

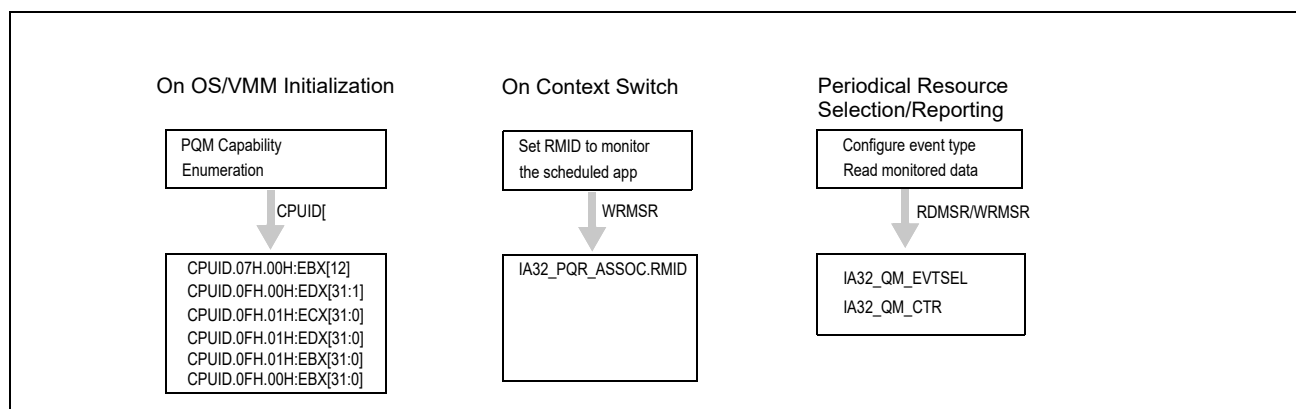


Figure 20-19. Platform Shared Resource Monitoring Usage Flow

20.18.3 Enumeration and Detecting Support of Cache Monitoring Technology and Memory Bandwidth Monitoring

Software can query processor support of shared resource monitoring features capabilities by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.07H.00H:EBX.RDT_M[12] reports 1, the processor provides the following programming interfaces for shared resource monitoring, including Cache Monitoring Technology:

- CPUID.0FH (Shared Resource Monitoring Enumeration leaf) provides information on available resource types (see Section 20.18.4), and monitoring capabilities for each resource type (see Section 20.18.5). Note CMT and MBM capabilities are enumerated as separate event vectors using shared enumeration infrastructure under a given resource type.
- IA32_PQR_ASSOC.RMID: The per-logical-processor MSR, IA32_PQR_ASSOC, that OS/VMM can use to assign an RMID to each logical processor, see Section 20.18.6.
- IA32_QM_EVTSEL: This MSR specifies an Event ID (EvtID) and an RMID which the platform uses to look up and provide monitoring data in the monitoring counter, IA32_QM_CTR, see Section 20.18.7.
- IA32_QM_CTR: This MSR reports monitored resource data when available along with bits to allow software to check for error conditions and verify data validity.

Software must follow the following sequence of enumeration to discover Cache Monitoring Technology capabilities:

1. Use CPUID.00H to discover the "cpuid_maxLeaf" supported in the processor;
2. If cpuid_maxLeaf >= 7, then verify CPUID.07H.00H:EBX.RDT_M[12] is set;
3. If CPUID.07H.00H:EBX.RDT_M[12] = 1, then use CPUID.0FH.00H to query available resource types that support monitoring;
4. If CPUID.0FH.00H:EDX.CMT_L3[1] = 1, then use CPUID.0FH.01H to query the specific capabilities of L3 Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring.
5. If CPUID.0FH.00H:EDX reports additional resource types supporting monitoring, then use CPUID.0FH.ResID for a corresponding resource type ID (ResID) as enumerated by the bit position of CPUID.0FH.00H:EDX.

20.18.4 Monitoring Resource Type and Capability Enumeration

CPUID.0FH (Shared Resource Monitoring Enumeration leaf) provides one sub-leaf (sub-function 0) that reports shared enumeration infrastructure, and one or more sub-functions that report feature-specific enumeration data:

- Monitoring leaf sub-function 0 enumerates available resources that support monitoring, i.e., use CPUID.0FH.00H. In the initial implementation, L3 cache is the only resource type available. Each supported resource type is represented by a bit in CPUID.0FH.00H:EDX[31:1]. The bit position corresponds to the sub-leaf index (ResID) that software must use to query details of the monitoring capability of that resource type (see Figure 20-21 and Figure 20-22). Reserved bits of CPUID.0FH.00H:EDX[31:2] correspond to unsupported

sub-leaves of the CPUID.0FH leaf. Additionally, CPUID.0FH.00H:EBX reports the highest RMID value of any resource type that supports monitoring in the processor.

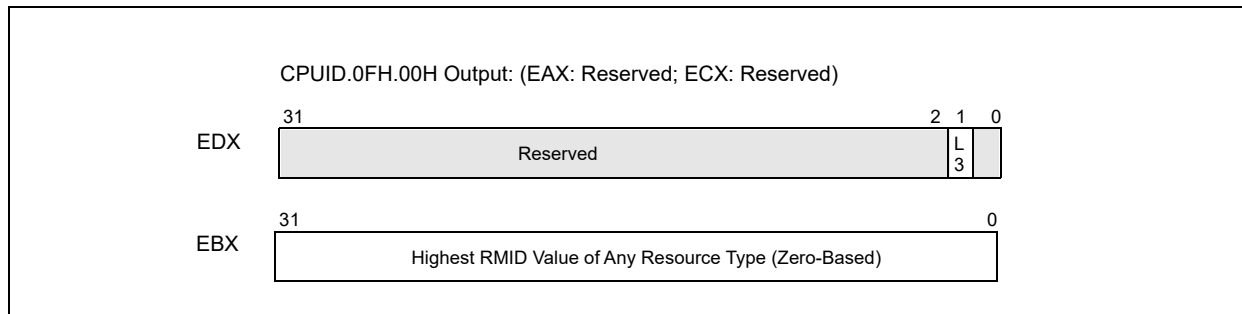


Figure 20-20. CPUID.0FH.00H Monitoring Resource Type Enumeration

20.18.5 Feature-Specific Enumeration

Each additional sub-leaf of CPUID.0FH.ResID enumerates the specific details for software to program monitoring MSRs using the resource type associated with the given ResID.

Note that in future Monitoring implementations the meanings of the returned registers may vary in other sub-leaves that are not yet defined. The registers will be specified and defined on a per-ResID basis.

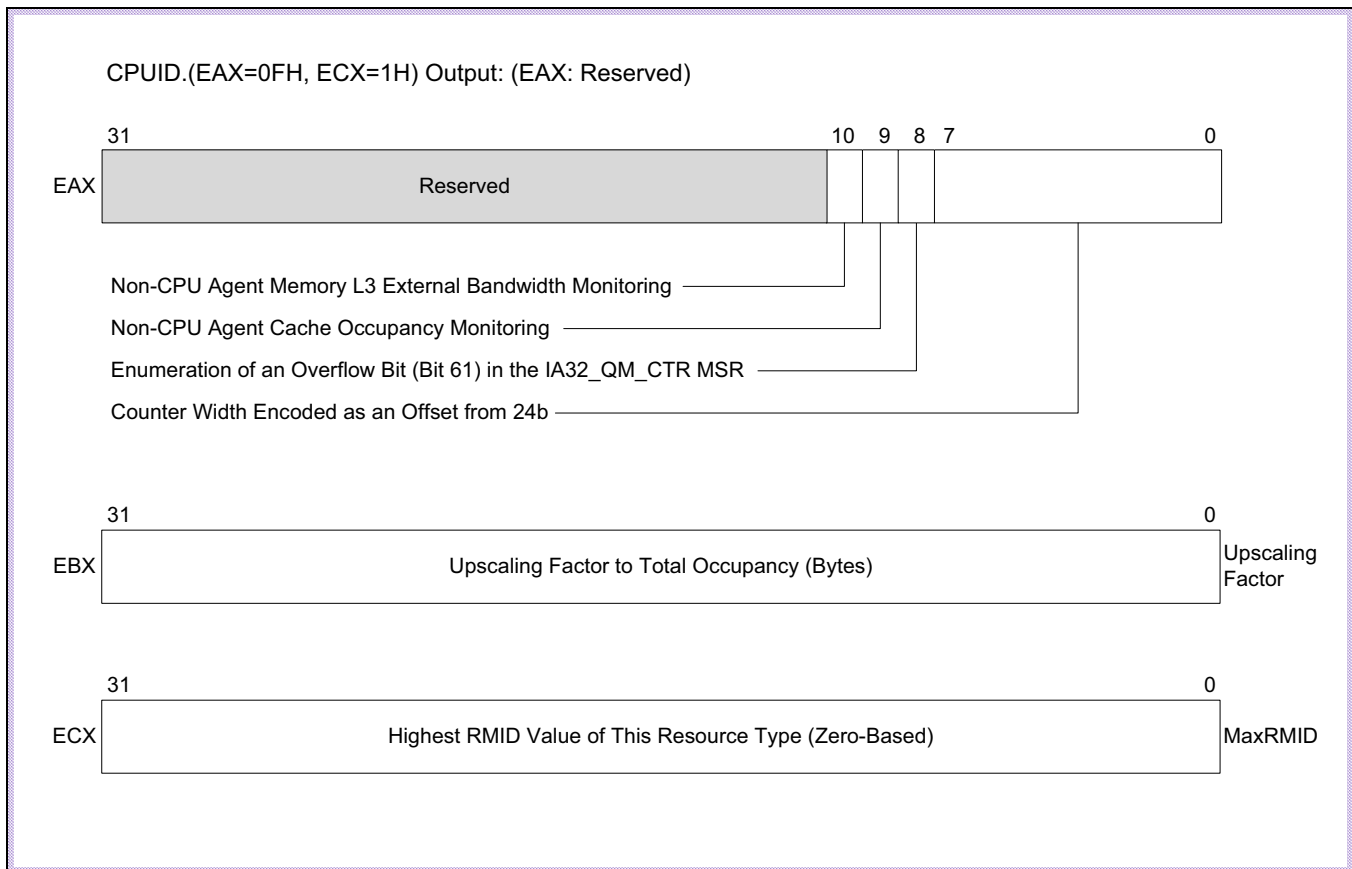


Figure 20-21. L3 Cache Monitoring Capability Enumeration Data (CPUID.0FH.01H)

CPUID.0FH.01H:EAX[7:0]: Encode counter width as offset from 24b. See Section 20.18.5.2 for details. Bits 31:11 of EAX are reserved.

CPUID.0FH.01H:EAX[8]: If 1, indicates the presence of an overflow bit in the IA32_QM_CTR MSR. See Section 20.18.5.2 for details. Bits 31:11 of EAX are reserved.

CPUID.0FH.01H:EAX[9]: If 1, indicates the presence of non-CPU agent Intel RDT CMT support. See Section 20.20 for details. Bits 31:11 of EAX are reserved.

CPUID.0FH.01H:EAX[10]: If 1, indicates the presence of non-CPU agent Intel RDT MBM support. See Section 20.20 for details. Bits 31:11 of EAX are reserved.

For each supported Cache Monitoring resource type, hardware supports only a finite number of RMIDs.

CPUID.0FH.01H:ECX enumerates the highest RMID value that can be monitored with this resource type, see Figure 20-21.

CPUID.0FH.01H:EDX specifies a bit vector that is used to look up the EventID (See Figure 20-22 and Table 20-18) that software must program with IA32_QM_EVTSEL in order to retrieve event data. After software configures IA32_QMEVTSEL with the desired RMID and EventID, it can read the resulting data from IA32_QM_CTR. The raw numerical value reported from IA32_QM_CTR can be converted to the final value (occupancy in bytes or bandwidth in bytes per sampled time period) by multiplying the counter value by the value from CPUID.0FH.01H:EBX, see Figure 20-21.

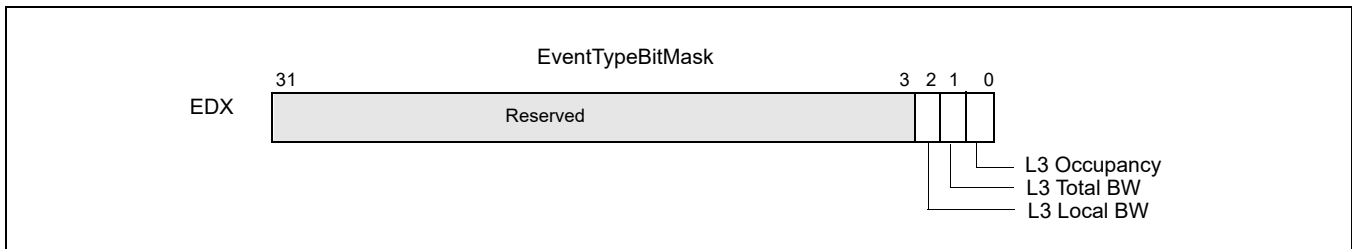


Figure 20-22. L3 Cache Monitoring Capability Enumeration Event Type Bit Vector (CPUID.0FH.01H)

20.18.5.1 Cache Monitoring Technology

On processors for which Cache Monitoring Technology supports the L3 cache occupancy event, CPUID.0FH.01H:EDX returns with bit 0 set. The corresponding event ID is shown in Table 20-18. The L3 occupancy data accumulated in the IA32_QM_CTR MSR can be converted to total occupancy (in bytes) by multiplying with CPUID.0FH.01H:EBX.

Event codes for Cache Monitoring Technology are discussed in the next section.

20.18.5.2 Memory Bandwidth Monitoring

On processors that monitoring supports Memory Bandwidth Monitoring using ResID=1 (L3), two additional bits are defined in the vector at CPUID.0FH.01H:EDX:

- CPUID.0FH.01H:EDX[1]: indicates the L3 total external bandwidth monitoring event is supported if set. This event monitors the L3 total external bandwidth to the next level of the cache hierarchy, including all demand and prefetch misses from the L3 to the next hierarchy of the memory system. In most platforms, this represents memory bandwidth.
- CPUID.0FH.01H:EDX[2]: indicates L3 local memory bandwidth monitoring event is supported if set. This event monitors the L3 external bandwidth satisfied by the local memory. In most platforms that support this event, L3 requests are likely serviced by a memory system with non-uniform memory architecture. This allows bandwidth to off-package memory resources to be tracked by subtracting local from total bandwidth (for instance, bandwidth over QPI to a memory controller on another physical processor could be tracked by subtraction). Note that it is not possible to read the local and total bandwidth atomically; multiple operations are needed. Because of this, it is possible for the counters to change in between the two reads.

The corresponding Event ID is shown in Table 20-18. The L3 bandwidth data accumulated in IA32_QM_CTR can be converted to total bandwidth (in bytes) using CPUID.0FH.01H:EBX.

Table 20-18. Monitoring Supported Event IDs

Event Type	Event ID	Context
L3 Cache Occupancy	01H	Cache Monitoring Technology
L3 Total External Bandwidth	02H	MBM
L3 Local External Bandwidth	03H	MBM
Reserved	All other event codes	N/A

A field is added to CPUID to enumerate the MBM counter width in platforms that support the extensible MBM counter width feature.

- CPUID.0FH.01H:EAX[7:0]: Encode counter width as offset from 24b in bits[7:0]. In EAX bits 7:0, the counter width is encoded as an offset from 24b. A value of zero in this field means 24-bit counters are supported. A value of 8 indicates that 32-bit counters are supported, as in the 3rd generation Intel Xeon Scalable Processor Family. With this enumerable counter width, the requirement that software polls at 1Hz is removed. Software may poll at a varying rate with a reduced risk of rollover. Under typical conditions, rollover will likely require hundreds of seconds (though this value is not explicitly specified and may vary and decrease in future processor generations as memory bandwidths increase). Suppose software seeks to ensure that rollover does not occur more than once between samples. In that case, sampling at 1Hz while consuming the enumerated counter widths' worth of data will provide this guarantee for a specific platform and counter width under all conditions.
- CPUID.0FH.01H:EAX[8]: Enumeration of the presence of an overflow bit in the IA32_QM_CTR MSR via EAX bit[8]. Software that uses the MBM event retrieval MSR interface should be updated to comprehend this new format, which enables up to 62-bit MBM counters to be provided by future platforms. Higher-level software that consumes the resulting bandwidth values is not expected to be affected. An overflow bit is defined in the IA32_QM_CTR MSR, bit 61, if CPUID.0FH.01H:EAX[8] is set. This rollover bit will be set on overflow of the MBM counters and reset upon read. Current processors do not support this capability.

20.18.6 Monitoring Resource RMID Association

After Monitoring and sub-features have been enumerated, software can begin using the monitoring features. The first step is to associate a given software thread (or multiple threads as part of an application, VM, group of applications or other abstraction) with an RMID.

Note that the process of associating an RMID with a given software thread is the same for all shared resource monitoring features (CMT, MBM), and a given RMID number has the same meaning from the viewpoint of any logical processors in a package. Stated another way, a thread may be associated in a 1:1 mapping with an RMID, and that RMID may allow cache occupancy, memory bandwidth information or other monitoring data to be read back later with monitoring event codes (retrieving data is discussed in a previous section).

The association of an application thread with an RMID requires an OS to program the per-logical-processor MSR IA32_PQR_ASSOC at context swap time (updates may also be made at any other arbitrary points during program execution such as application phase changes). The IA32_PQR_ASSOC MSR specifies the active RMID that monitoring hardware will use to tag internal operations, such as L3 cache requests. The layout of the MSR is shown in Figure 20-23. Software specifies the active RMID to monitor in the IA32_PQR_ASSOC.RMID field. The width of the RMID field can vary from one implementation to another, and is derived from $\text{Ceil}(\text{LOG}_2(1 + \text{CPUID.0FH.00H:EBX}[31:0]))$. The value of IA32_PQR_ASSOC after power-on is 0.

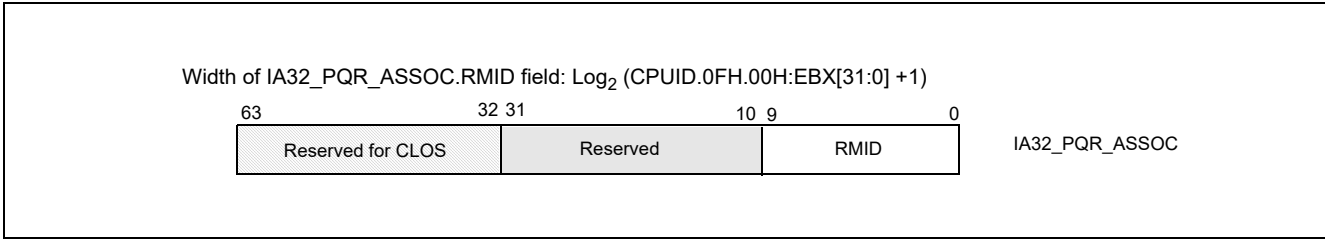


Figure 20-23. IA32_PQR_ASSOC MSR

In the initial implementation, the width of the RMID field is up to 10 bits wide, zero-referenced and fully encoded. However, software must use CPUID to query the maximum RMID supported by the processor. If a value larger than the maximum RMID is written to IA32_PQR_ASSOC.RMID, a #GP(0) fault will be generated.

RMIDs have a global scope within the physical package- if an RMID is assigned to one logical processor then the same RMID can be used to read multiple thread attributes later (for example, L3 cache occupancy or external bandwidth from the L3 to the next level of the cache hierarchy). In a multiple LLC platform the RMIDs are to be reassigned by the OS or VMM scheduler when an application is migrated across LLCs.

Note that in a situation where Monitoring supports multiple resource types, some upper range of RMIDs (e.g., RMID 31) may only be supported by one resource type but not by another resource type.

20.18.7 Monitoring Resource Selection and Reporting Infrastructure

The reporting mechanism for Cache Monitoring Technology and other related features is architecturally exposed as an MSR pair that can be programmed and read to measure various metrics such as the L3 cache occupancy (CMT) and bandwidths (MBM) depending on the level of Monitoring support provided by the platform. Data is reported back on a per-RMID basis. These events do not trigger based on event counts or trigger APIC interrupts (e.g., no Performance Monitoring Interrupt occurs based on counts). Rather, they are used to sample counts explicitly.

The MSR pair for the shared resource monitoring features (CMT, MBM) is separate from and not shared with architectural PerfMon counters, meaning software can use these monitoring features simultaneously with the PerfMon counters.

Access to the aggregated monitoring information is accomplished through the following programmable monitoring MSRs:

- IA32_QM_EVTSEL: This MSR provides a role similar to the event select MSRs for programmable performance monitoring described in Chapter 18. The simplified layout of the MSR is shown in Figure 20-24. IA32_QM_EVTSEL.EvtID (bits 7:0) specifies an event code of a supported resource type for hardware to report monitored data associated with IA32_QM_EVTSEL.RMID (bits 41:32). Software can configure IA32_QM_EVTSEL.RMID with any RMID that is active within the physical processor. The width of IA32_QM_EVTSEL.RMID matches that of IA32_PQR_ASSOC.RMID. Supported event codes for the IA32_QM_EVTSEL register are shown in Table 20-18. Note that valid event codes may not necessarily map directly to the bit position used to enumerate support for the resource via CPUID.

Software can program an RMID / Event ID pair into the IA32_QM_EVTSEL MSR bit field to select an RMID to read a particular counter for a given resource. The currently supported list of Monitoring Event IDs is discussed in Section 20.18.5, which covers feature-specific details.

Thread access to the IA32_QM_EVTSEL and IA32_QM_CTR MSR pair should be serialized (that is, treated as a critical section under lock) to avoid situations where one thread changes the RMID/EvtID just before another thread reads monitoring data from IA32_QM_CTR.
- IA32_QM_CTR: This MSR reports monitored data when available. It contains three bit fields. If software configures an unsupported RMID or event type in IA32_QM_EVTSEL, then IA32_QM_CTR.Error (bit 63) will be set, indicating there is no valid data to report. If IA32_QM_CTR.Unavailable (bit 62) is set, it indicates monitored data for the RMID is not available, and IA32_QM_CTR.data (bits 61:0) should be ignored. Therefore, IA32_QM_CTR.data (bits 61:0) is valid only if bits 63 and 62 are both clear. The IA32_QM_CTR.Overflow (bit 61) is present if CPUID.0FH.01H:EAX[8] is set. This bit is set on overflow of the MBM counters and will reset

upon read. For Cache Monitoring Technology, software can convert IA32_QM_CTR.data into cache occupancy or bandwidth metrics expressed in bytes by multiplying with the conversion factor from CPUID.0FH.01H:EBX.

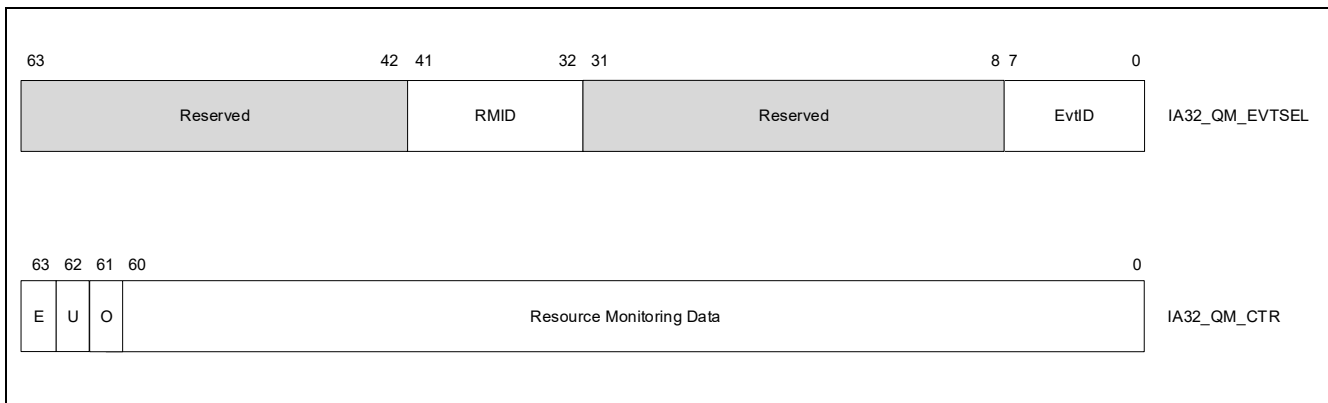


Figure 20-24. IA32_QM_EVTSEL and IA32_QM_CTR MSRs

20.18.8 Monitoring Programming Considerations

Figure 20-25 illustrates how system software can program IA32_QM_EVTSEL and IA32_QM_CTR to perform resource monitoring.

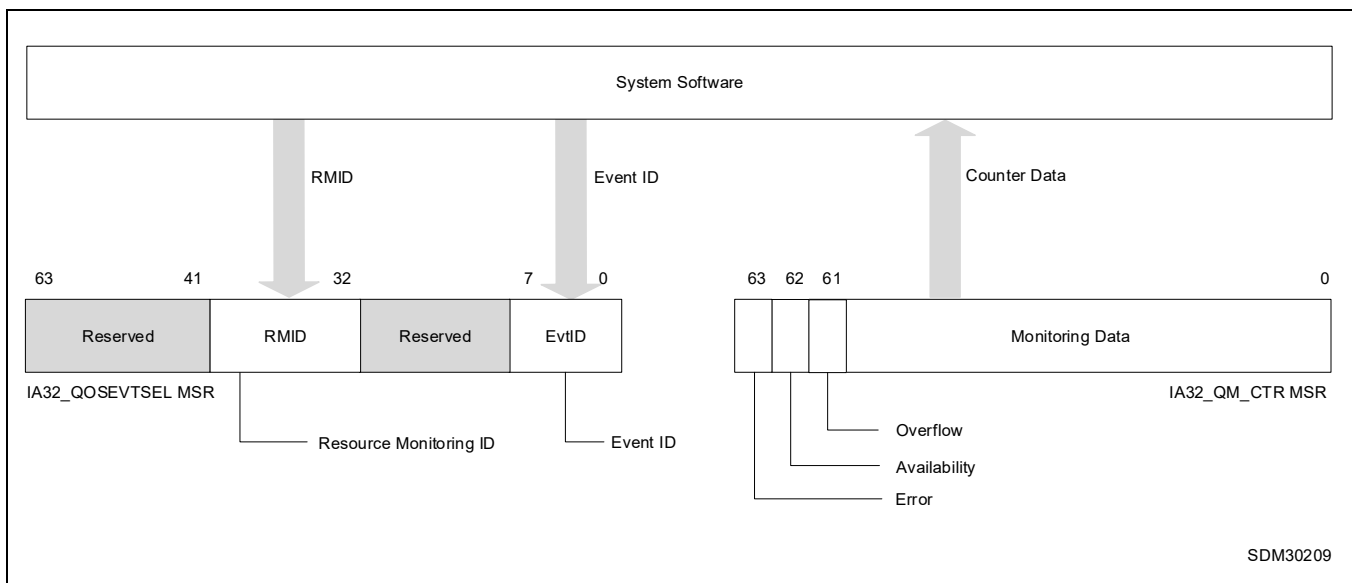


Figure 20-25. Software Usage of Cache Monitoring Resources

Though the field provided in IA32_QM_CTR allows for up to 62 bits of data to be returned, often a subset of bits are used. With Cache Monitoring Technology for instance, the number of bits used is the base-two logarithm of the total cache size divided by the Upscaling Factor from CPUID.

In Memory Bandwidth Monitoring, the initial counter size is 24 bits, and retrieving the value at 1Hz or faster is sufficient to ensure at most one rollover per sampling period. Any changes to counter width are enumerated to software; see Section 20.18.5.2 for details.

20.18.8.1 Monitoring Dynamic Configuration

Both the IA32_QM_EVTSEL and IA32_PQR_ASSOC registers are accessible and modifiable at any time during execution using RDMSR/WRMSR unless otherwise noted. When writing to these MSRs a #GP(0) will be generated if any of the following conditions occur:

- A reserved bit is modified,
- An RMID exceeding the maximum RMID is used.

20.18.8.2 Monitoring Operation With Power Saving Features

Some advanced power management features such as deep package C-states may shrink the L3 cache and cause CMT occupancy count to be reduced. MBM bandwidth counts may increase due to flushing cached data out of L3.

20.18.8.3 Monitoring Operation with Other Operating Modes

The states in IA32_PQR_ASSOC and monitoring counter are unmodified across an SMI delivery. Thus, the execution of SMM handler code and SMM handler's data can manifest as spurious contribution in the monitored data.

It is possible for an SMM handler to minimize the impact on of spurious contribution in the QOS monitoring counters by reserving a dedicated RMID for monitoring the SMM handler. Such an SMM handler can save the previously configured QOS Monitoring state immediately upon entering SMM, and restoring the QOS monitoring state back to the prev-SMM RMID upon exit.

20.18.8.4 Monitoring Operation with RAS Features

In general, the Reliability, Availability, and Serviceability (RAS) features present in Intel Platforms are not expected to significantly affect shared resource monitoring counts. In cases where software RAS features cause memory copies or cache accesses, these may be tracked and may influence the shared resource monitoring counter values.

20.19 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) ALLOCATION FEATURES

The Intel Resource Director Technology (Intel RDT) feature set provides a set of allocation (resource control) capabilities including Cache Allocation Technology (CAT) and Code and Data Prioritization (CDP). The Intel Xeon processor E5 v4 family (and a subset of communication-focused processors in the Intel Xeon E5 v3 family) introduce capabilities to configure and make use of the Cache Allocation Technology (CAT) mechanisms on the L3 cache. Certain Intel Atom processors also provide support for control over the L2 cache, with capabilities as described below. The programming interface for Cache Allocation Technology and for the more general allocation capabilities are described in the rest of this chapter. The CAT and CDP capabilities, where architecturally supported, may be detected and enumerated in software using the CPUID instruction, as described in this chapter.

The Intel Xeon Scalable Processor Family introduces the Memory Bandwidth Allocation (MBA) feature which provides indirect control over the memory bandwidth available to CPU cores, and is discussed later in this chapter.

20.19.1 Introduction to Cache Allocation Technology (CAT)

Cache Allocation Technology enables an Operating System (OS), Hypervisor /Virtual Machine Manager (VMM) or similar system service management agent to specify the amount of cache space into which an application can fill (as a hint to hardware - certain features such as power management may override CAT settings). Specialized user-level implementations with minimal OS support are also possible, though not necessarily recommended (see notes below for OS/Hypervisor with respect to ring 3 software and virtual guests). Depending on the processor family, L2 or L3 cache allocation capability may be provided, and the technology is designed to scale across multiple cache levels and technology generations.

Software can determine which levels are supported in a given platform programmatically using CPUID as described in the following sections.

The CAT mechanisms defined in this document provide the following key features:

- A mechanism to enumerate platform Cache Allocation Technology capabilities and available resource types that provides CAT control capabilities. For implementations that support Cache Allocation Technology, CPUID provides enumeration support to query which levels of the cache hierarchy are supported and specific CAT capabilities, such as the max allocation bitmask size.
- A mechanism for the OS or Hypervisor to configure the amount of a resource available to a particular Class of Service via a list of allocation bitmasks.
- Mechanisms for the OS or Hypervisor to signal the Class of Service to which an application belongs.
- Hardware mechanisms to guide the LLC fill policy when an application has been designated to belong to a specific Class of Service.

Note that for many usages, an OS or Hypervisor may not want to expose Cache Allocation Technology mechanisms to Ring3 software or virtualized guests.

The Cache Allocation Technology feature enables more cache resources (i.e., cache space) to be made available for high priority applications based on guidance from the execution environment as shown in Figure 20-26. The architecture also allows dynamic resource reassignment during runtime to further optimize the performance of the high priority application with minimal degradation to the low priority app. Additionally, resources can be rebalanced for system throughput benefit across uses cases of OSes, VMMs, containers, and other scenarios by managing the CPUID and MSR interfaces. This section describes the hardware and software support required in the platform including what is required of the execution environment (i.e., OS/VMM) to support such resource control. Note that in Figure 20-26 the L3 Cache is shown as an example resource.

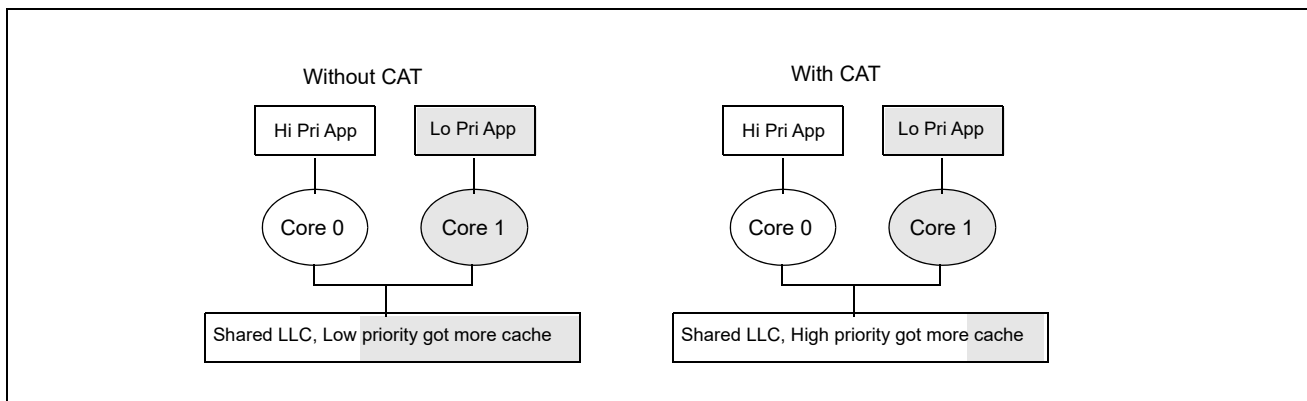


Figure 20-26. Cache Allocation Technology Enables Allocation of More Resources to High Priority Applications

20.19.2 Cache Allocation Technology Architecture

The fundamental goal of Cache Allocation Technology is to enable resource allocation based on application priority or Class of Service (CLOS). The processor exposes a set of Classes of Service into which applications (or individual threads) can be assigned. Cache allocation for the respective applications or threads is then restricted based on the class with which they are associated. Each Class of Service can be configured using capacity bitmasks (CBMs) which represent capacity and indicate the degree of overlap and isolation between classes. For each logical processor there is a register exposed (referred to here as the IA32_PQR_ASSOC MSR or PQR) to allow the OS/VMM to specify a CLOS when an application, thread or VM is scheduled.

The usage of Classes of Service (CLOS) are consistent across resources and a CLOS may have multiple resource control attributes attached, which reduces software overhead at context swap time. Rather than adding new types of CLOS tags per resource for instance, the CLOS management overhead is constant. Cache allocation for the indicated application/thread/container/VM is then controlled automatically by the hardware based on the class and the bitmask associated with that class. Bitmasks are configured via the IA32_resourceType_MASK_n MSRs, where resourceType indicates a resource type (e.g., "L3" for the L3 cache) and "n" indicates a CLOS number.

The basic ingredients of Cache Allocation Technology are as follows:

- An architecturally exposed mechanism using CPUID to indicate whether CAT is supported, and what resource types are available which can be controlled.

- For each available resourceType, CPUID also enumerates the total number of Classes of Services and the length of the capacity bitmasks that can be used to enforce cache allocation to applications on the platform.
- An architecturally exposed mechanism to allow the execution environment (OS/VMM) to configure the behavior of different classes of service using the bitmasks available.
- An architecturally exposed mechanism to allow the execution environment (OS/VMM) to assign a CLOS to an executing software thread (i.e., associating the active CR3 of a logical processor with the CLOS in IA32_PQR_ASSOC).
- Implementation-dependent mechanisms to indicate which CLOS is associated with a memory access and to enforce the cache allocation on a per CLOS basis.

A capacity bitmask (CBM) provides a hint to the hardware indicating the cache space an application should be limited to as well as providing an indication of overlap and isolation in the CAT-capable cache from other applications contending for the cache. The bit length of the capacity mask available generally depends on the configuration of the cache and is specified in the enumeration process for CAT in CPUID (this may vary between models in a processor family as well). Similarly, other parameters such as the number of supported CLOS may vary for each resource type, and these details can be enumerated via CPUID.

	M7	M6	M5	M4	M3	M2	M1	M0	
CLOS0	A	A	A	A	A	A	A	A	Default Bitmask
CLOS1	A	A	A	A	A	A	A	A	
CLOS2	A	A	A	A	A	A	A	A	
CLOS3	A	A	A	A	A	A	A	A	
	M7	M6	M5	M4	M3	M2	M1	M0	
CLOS0	A	A	A	A	A	A	A	A	Overlapped Bitmask
CLOS1					A	A	A	A	
CLOS2							A	A	
CLOS3								A	
	M7	M6	M5	M4	M3	M2	M1	M0	
CLOS0	A	A	A	A					Isolated Bitmask
CLOS1					A	A			
CLOS2							A		
CLOS3								A	

Figure 20-27. Examples of Cache Capacity Bitmasks

Sample cache capacity bitmasks for a bit length of 8 are shown in Figure 20-27. Note that all (and only) contiguous '1' combinations are allowed (e.g., FFFFH, 0FF0H, 003CH, etc.), unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type. Attempts to program a value without contiguous '1's (including zero) will result in a general protection fault (#GP(0)). It is generally expected that in way-based implementations, one capacity mask bit corresponds to some number of ways in cache, but the specific mapping is implementation-dependent. In all cases, a mask bit set to '1' specifies that a particular Class of Service can allocate into the cache subset represented by that bit. A value of '0' in a mask bit specifies that a Class of Service cannot

allocate into the given cache subset. In general, allocating more cache to a given application is usually beneficial to its performance.

Figure 20-27 also shows three examples of sets of Cache Capacity Bitmasks. For simplicity these are represented as 8-bit vectors, though this may vary depending on the implementation and how the mask is mapped to the available cache capacity. The first example shows the default case where all 4 Classes of Service (the total number of CLOS are implementation-dependent) have full access to the cache. The second case shows an overlapped case, which would allow some lower-priority threads to share cache space with the highest priority threads. The third case shows various non-overlapped partitioning schemes. As a matter of software policy for extensibility, CLOS0 should typically be considered and configured as the highest priority CLOS, followed by CLOS1, and so on, though there is no hardware restriction enforcing this mapping. When the system boots all threads are initialized to CLOS0, which has full access to the cache by default.

Though the representation of the CBMs looks similar to a way-based mapping they are independent of any specific enforcement implementation (e.g., way partitioning.) Rather, this is a convenient manner to represent capacity, overlap, and isolation of cache space. For example, executing a POPCNT instruction (population count of set bits) on the capacity bitmask can provide the fraction of cache space that a class of service can allocate into. In addition to the fraction, the exact location of the bits also shows whether the class of service overlaps with other classes of service or is entirely isolated in terms of cache space used.

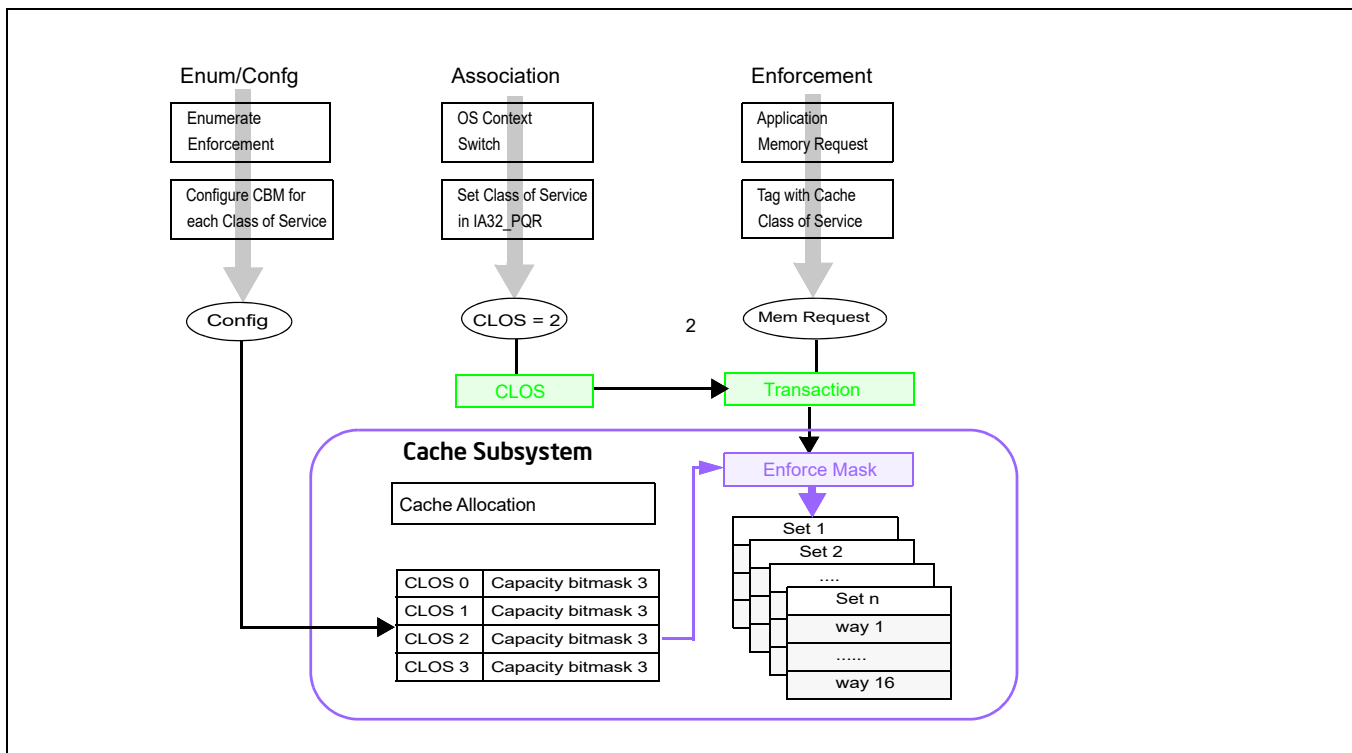


Figure 20-28. Class of Service and Cache Capacity Bitmasks

Figure 20-28 shows how the Cache Capacity Bitmasks and the per-logical-processor Class of Service are logically used to enable Cache Allocation Technology. All (and only) contiguous 1's in the CBM are permitted, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type. The length of a CBM may vary from resource to resource or between processor generations and can be enumerated using CPUID. From the available mask set and based on the goals of the OS/VMM (shared or isolated cache, etc.) bitmasks are selected and associated with different classes of service. For the available Classes of Service the associated CBMs can be programmed via the global set of CAT configuration registers (in the case of L3 CAT, via the IA32_L3_MASK_n MSRs, where "n" is the Class of Service, starting from zero). In all architectural implementations supporting CPUID it is possible to change the CBMs dynamically, during program execution, unless stated otherwise by Intel.

The currently running application's Class of Service is communicated to the hardware through the per-logical-processor PQR MSR (IA32_PQR_ASSOC MSR). When the OS schedules an application thread on a logical processor, the application thread is associated with a specific CLOS (i.e., the corresponding CLOS in the PQR) and all requests to the CAT-capable resource from that logical processor are tagged with that CLOS (in other words, the application thread is configured to belong to a specific CLOS). The cache subsystem uses this tagged request information to enforce QoS. The capacity bitmask may be mapped into a way bitmask (or a similar enforcement entity based on the implementation) at the cache before it is applied to the allocation policy. For example, the capacity bitmask can be an 8-bit mask and the enforcement may be accomplished using a 16-way bitmask for a cache enforcement implementation based on way partitioning.

The following sections describe extensions of CAT such as Code and Data Prioritization (CDP), followed by details on specific features such as L3 CAT, L3 CDP, L2 CAT, and L2 CDP. Depending on the specific processor a mix of features may be supported, and CPUID provides enumeration capabilities to enable software to dynamically detect the set of supported features.

20.19.3 Code and Data Prioritization (CDP) Technology

Code and Data Prioritization Technology is an extension of CAT. CDP enables isolation and separate prioritization of code and data fetches to the L2 or L3 cache in a software configurable manner, depending on hardware support, which can enable workload prioritization and tuning of cache capacity to the characteristics of the workload. CDP extends Cache Allocation Technology (CAT) by providing separate code and data masks per Class of Service (CLOS). Support for the L2 CDP feature and the L3 CDP features are separately enumerated (via CPUID) and separately controlled (via remapping the L2 CAT MSRs or L3 CAT MSRs respectively). Section 20.19.6.3 and Section 20.19.7 provide details on enumerating, controlling, and enabling L3 and L2 CDP respectively, while this section provides a general overview.

The L3 CDP feature was first introduced on the Intel Xeon E5 v4 family of server processors, as an extension to L3 CAT. The L2 CDP feature is first introduced on future Intel Atom family processors, as an extension to L2 CAT.

By default, CDP is disabled on the processor. If the CAT MSRs are used without enabling CDP, the processor operates in a traditional CAT-only mode. When CDP is enabled:

- The CAT mask MSRs are re-mapped into interleaved pairs of mask MSRs for data or code fetches (see Figure 20-29).
- The range of CLOS for CAT is re-indexed, with the lower-half of the CLOS range available for CDP.

Using the CDP feature, virtual isolation between code and data can be configured on the L2 or L3 cache if desired, similar to how some processor cache levels provide separate L1 data and L1 instruction caches.

Like the CAT feature, CDP may be dynamically configured by privileged software at any point during normal system operation, including dynamically enabling or disabling the feature provided that certain software configuration requirements are met (see Section 20.19.5).

An example of the operating mode of CDP is shown in Figure 20-29. Shown at the top are traditional CAT usage models where capacity masks map 1:1 with a CLOS number to enable control over the cache space which a given CLOS (and thus applications, threads or VMs) may occupy. Shown at the bottom are example mask configurations where CDP is enabled, and each CLOS number maps 1:2 to two masks, one for code and one for data. This enables code and data to be either overlapped or isolated to varying degrees either globally or on a per-CLOS basis, depending on application and system needs.

Example of CAT-Only Usage - 16 bit Capacity Masks															
CLOS0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
CLOS1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
CLOS2	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
CLOS3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1

Traditional CAT

Example of Code/Data Prioritization Usage - 16 bit Capacity Masks															
CLOS0.Data	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
CLOS0.Code	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
CLOS1.Data	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0
CLOS1.Code	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
Other CLOS.Data	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
Other CLOS.Code	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

CAT with CDP

Figure 20-29. Code and Data Capacity Bitmasks of CDP

When CDP is enabled, the existing mask space for CAT-only operation is split. As an example if the system supports 16 CAT-only CLOS, when CDP is enabled the same MSR interfaces are used, however half of the masks correspond to code, half correspond to data, and the effective number of CLOS is reduced by half. Code/Data masks are defined per-CLOS and interleaved in the MSR space as described in subsequent sections.

In cases where CPUID exposes a non-even number of supported Classes of Service for the CAT or CDP features, software using CDP should use the lower matched pairs of code/data masks, and any upper unpaired masks should not be used. As an example, if CPUID exposes 5 CLOS, when CDP is enabled then two code/data pairs are available (masks 0/1 for CLOS[0] data/code and masks 2/3 for CLOS[1] data/code), however the upper un-paired mask should not be used (mask 4 in this case) or undefined behavior may result.

20.19.4 Enabling Cache Allocation Technology Usage Flow

Figure 20-30 illustrates the key steps for OS/VMM to detect support of Cache Allocation Technology and enable priority-based resource allocation for a CAT-capable resource.

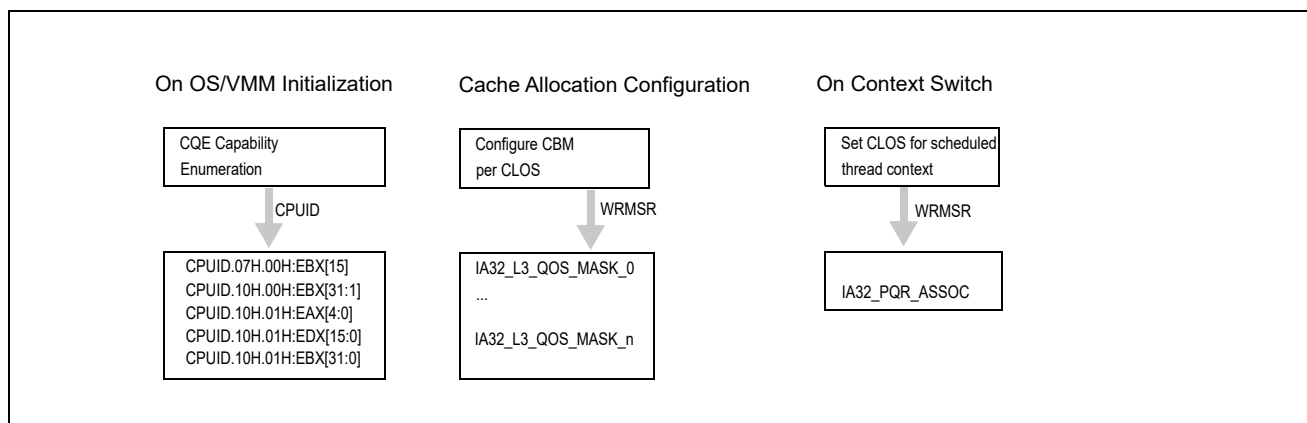


Figure 20-30. Cache Allocation Technology Usage Flow

Enumeration and configuration of L2 CAT is similar to L3 CAT, however CPUID details and MSR addresses differ. Common CLOS are used across the features.

20.19.4.1 Enumeration and Detection Support of Cache Allocation Technology

Software can query processor support of CAT capabilities by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.07H.00H:EBX.RDT_A[15] reports 1, the processor supports software control over shared processor resources. Software must use CPUID.10H to enumerate additional details of available resource types, classes of services and capability bitmasks. The programming interfaces provided by Cache Allocation Technology include:

- CPUID.10H (Cache Allocation Technology Enumeration leaf) and its sub-functions provide information on available resource types, and CAT capability for each resource type (see Section 20.19.4.2).
- IA32_L3_MASK_n: A range of MSRs is provided for each resource type, each MSR within that range specifying a software-configured capacity bitmask for each class of service. For L3 with Cache Allocation support, the CBM is specified using one of the IA32_L3_QOS_MASK_n MSR, where 'n' corresponds to a number within the supported range of CLOS, i.e., the range between 0 and CPUID.10H.ResID:EDX[15:0], inclusive. See Section 20.19.4.3 for details.
- IA32_L2_MASK_n: A range of MSRs is provided for L2 Cache Allocation Technology, enabling software control over the amount of L2 cache available for each CLOS. Similar to L3 CAT, a CBM is specified for each CLOS using the set of registers, IA32_L2_QOS_MASK_n MSR, where 'n' ranges from zero to the maximum CLOS number reported for L2 CAT in CPUID. See Section 20.19.4.3 for details.

The L2 mask MSRs are scoped at the same level as the L2 cache (similarly, the L3 mask MSRs are scoped at the same level as the L3 cache). Software may determine which logical processors share an MSR (for instance local to a core, or shared across multiple cores) by performing a write to one of these MSRs and noting which logical threads observe the change. Example flows for a similar method to determine register scope are described in Section 18.5.2, “System Software Recommendation for Managing CMCI and Machine Check Resources.” Software may also use CPUID.04H to determine the maximum number of logical processor IDs that may share a given level of the cache.

- IA32_PQR_ASSOC.CLOS: The IA32_PQR_ASSOC MSR provides a CLOS field that OS/VMM can use to assign a logical processor to an available CLOS. The set of CLOS are common across all allocation features, meaning that multiple features may be supported in the same processor without additional software CLOS management overhead at context swap time. See Section 20.19.4.4 for details.

20.19.4.2 Cache Allocation Technology: Resource Type and Capability Enumeration

CPUID.10H (Cache Allocation Technology Enumeration leaf) provides two or more sub-functions:

- CAT Enumeration leaf sub-function 0 enumerates available resource types that support allocation control, i.e., by using CPUID.10H.00H. Each supported resource type is represented by a bit field in CPUID.10H.00H:EBX[31:1]. The bit position of each set bit corresponds to a Resource ID (ResID), for instance ResID=1 is used to indicate L3 CAT support, and ResID=2 indicates L2 CAT support. The ResID is also the sub-leaf index that software must use to query details of the CAT capability of that resource type (see Figure 20-31).

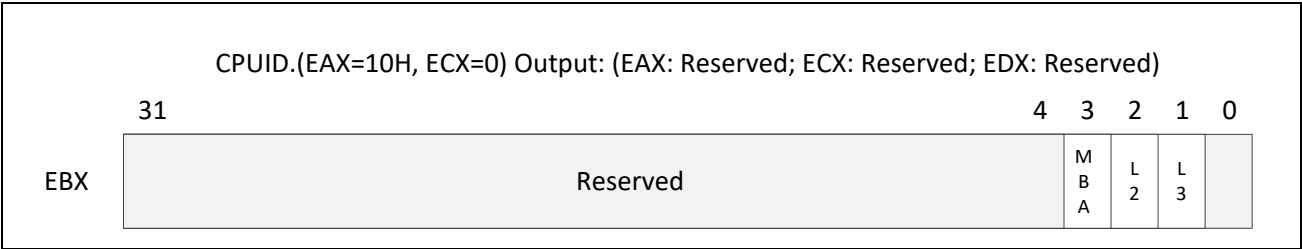


Figure 20-31. CPUID.10H.00H Available Resource Type Identification

- For ECX>0, EAX[4:0] reports the length of the capacity bitmask (ECX=1 or 2 for L3 CAT or L2 CAT respectively). Add one to the return value to get the result, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
- Sub-leaves of CPUID.10H with a non-zero ECX input matching a supported ResID enumerate the specific enforcement details of the corresponding ResID. The capabilities enumerated include the length of the capacity bitmasks and the number of Classes of Service for a given ResID. Software should query the capability of each available ResID that supports CAT from a sub-leaf of leaf 10H using the sub-leaf index reported by the corresponding non-zero bit in CPUID.10H.00H:EBX[31:1] in order to obtain additional feature details.

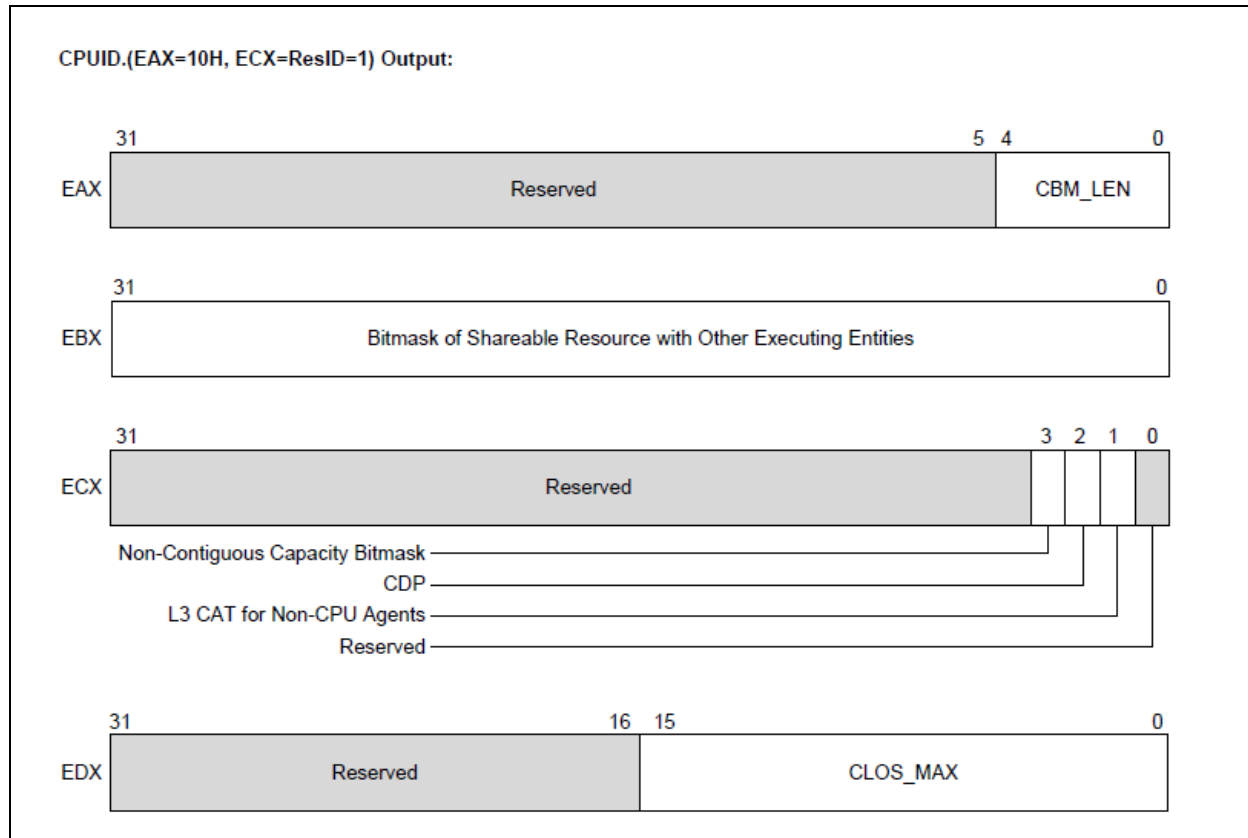


Figure 20-32. L3 Cache Allocation Technology and CDP Enumeration

- CAT capability for L3 is enumerated by CPUID.10H.01H, see Figure 20-32. The specific CAT capabilities reported by CPUID.10H.01H are:
 - CPUID.10H.ResID=1:EAX[4:0] reports the length of the capacity bitmask. Add one to the return value to get the result, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
 - CPUID.10H.01H:EBX[31:0] reports a bit mask. Each set bit within the length of the CBM indicates the corresponding unit of the L3 allocation may be used by other entities in the platform (e.g., an integrated graphics engine or hardware units outside the processor core and have direct access to L3). Each cleared bit within the length of the CBM indicates the corresponding allocation unit can be configured to implement a priority-based allocation scheme chosen by an OS/VMM without interference with other hardware agents in the system. Bits outside the length of the CBM are reserved.
 - CPUID.10H.01H:ECX[1]: If 1, indicates L3 CAT for non-CPU agents is supported. Bits 0 and 31:4 of ECX are reserved. See section 18.20 for details.
 - CPUID.10H.01H:ECX.CDP[2]: If 1, indicates L3 Code and Data Prioritization Technology is supported (see Section 20.19.5). Bits 0 and 31:4 of ECX are reserved.

- CPUID.10H.01H:ECX[3]: If 1, indicates non-contiguous capacity bitmask is supported. The bits that are set in the various IA32_L3_MASK_n registers do not have to be contiguous. Bits 0 and 31:4 of ECX are reserved.
- CPUID.10H.01H:EDX[15:0] reports the maximum CLOS supported for the resource (CLOS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported CLOS). Bits 31:16 are reserved.

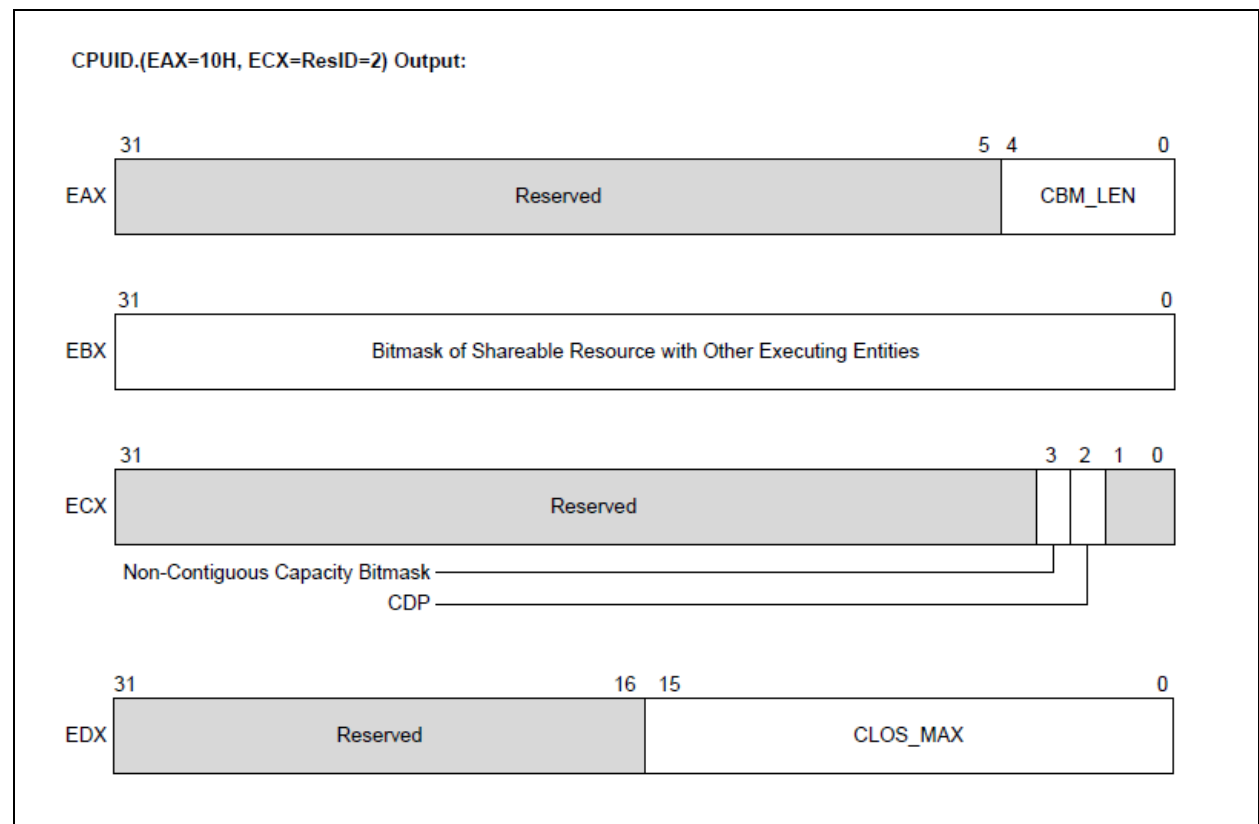


Figure 20-33. L2 Cache Allocation Technology

- CAT capability for L2 is enumerated by CPUID.10H.02H, see Figure 20-33. The specific CAT capabilities reported by CPUID.10H.02H are:
 - CPUID.10H.ResID=2:EAX[4:0] reports the length of the capacity bitmask. Add one to the return value to get the result, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
 - CPUID.10H.02H:EBX[31:0] reports a bit mask. Each set bit within the length of the CBM indicates the corresponding unit of the L2 allocation may be used by other entities in the platform. Each cleared bit within the length of the CBM indicates the corresponding allocation unit can be configured to implement a priority-based allocation scheme chosen by an OS/VMM without interference with other hardware agents in the system. Bits outside the length of the CBM are reserved.
 - CPUID.10H.02H:ECX.CDP[2]: If 1, indicates L2 Code and Data Prioritization Technology is supported (see Section 17.19.6). Bits 1:0 and 31:4 of ECX are reserved.
 - CPUID.10H.02H:ECX[3]: If 1, indicates non-contiguous capacity bitmask is supported. The bits which are set in the various IA32_L2_MASK_n registers do not have to be contiguous. Bits 1:0 and 31:4 of ECX are reserved.

- CPUID.10H.02H:EDX[15:0] reports the maximum CLOS supported for the resource (CLOS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported CLOS). Bits 31:16 are reserved.

A note on migration of Classes of Service (CLOS): Software should minimize migrations of CLOS across logical processors (across threads or cores), as a reduction in the performance of the Cache Allocation Technology feature may result if CLOS are migrated frequently. This is aligned with the industry-standard practice of minimizing unnecessary thread migrations across processor cores in order to avoid excessive time spent warming up processor caches after a migration. In general, for best performance, minimize thread migration and CLOS migration across processor logical threads and processor cores.

20.19.4.3 Cache Allocation Technology: Cache Mask Configuration

After determining the length of the capacity bitmasks (CBM) and number of CLOS supported using CPUID (see Section 20.19.4.2), each CLOS needs to be programmed with a CBM to dictate its available cache via a write to the corresponding IA32_resourceType_MASK_n register, where 'n' corresponds to a number within the supported range of CLOS, i.e., the range between 0 and CPUID.10H.ResID:EDX[15:0], inclusive, and 'resourceType' corresponds to a specific resource as enumerated by the set bits of CPUID.10H.00H:EBX[31:1], for instance, 'L2' or 'L3' cache.

A hierarchy of MSRs is reserved for Cache Allocation Technology registers of the form IA32_resourceType_MASK_n:

- From 0C90H through 0D8FH (inclusive), providing support for multiple sub-ranges to support varying resource types. The first supported resource type is 'L3', corresponding to the L3 cache in a platform. The MSRs range from 0C90H through 0D0FH (inclusive), enables support for up to 128 L3 CAT Classes of Service.

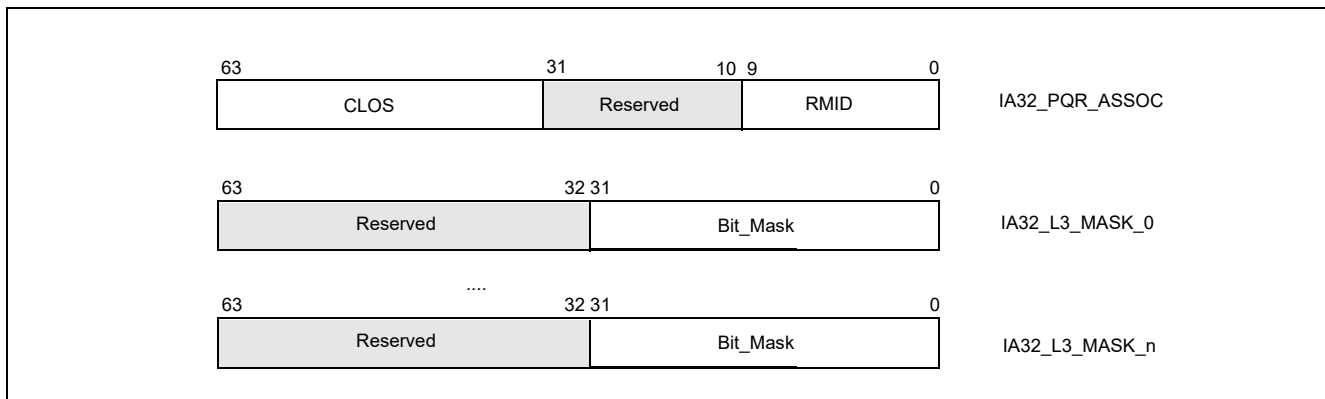


Figure 20-34. IA32_PQR_ASSOC, IA32_L3_MASK_n MSRs

- Within the same CAT range hierarchy, another set of registers is defined for resourceType 'L2', corresponding to the L2 cache in a platform, and MSRs IA32_L2_MASK_n are defined for n=[0,63] at addresses 0D10H through 0D4FH (inclusive).

Figure 20-34 and Figure 20-35 provide an overview of the relevant registers.

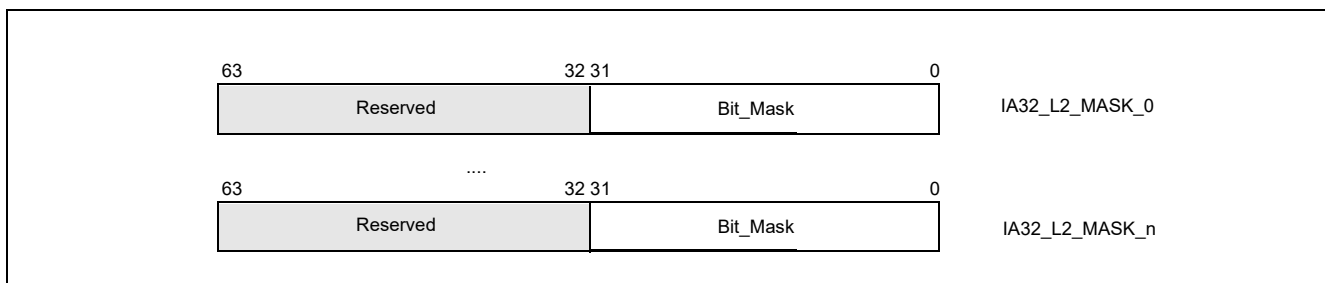


Figure 20-35. IA32_L2_MASK_n MSRs

All CAT configuration registers can be accessed using the standard RDMSR / WRMSR instructions. Note that once L3 or L2 CAT masks are configured, threads can be grouped into Classes of Service (CLOS) using the IA32_PQR_ASSOC MSR as described in Section 20.19.4.4, “Class of Service to Cache Mask Association: Common Across Allocation Features.”

20.19.4.4 Class of Service to Cache Mask Association: Common Across Allocation Features

After configuring the available classes of service with the preferred set of capacity bitmasks, the OS/VMM can set the IA32_PQR_ASSOC.CLOS of a logical processor to the class of service with the desired CBM when a thread context switch occurs. This allows the OS/VMM to indicate which class of service an executing thread/VM belongs within. Each logical processor contains an instance of the IA32_PQR_ASSOC register at MSR location 0C8FH, and Figure 20-34 shows the bit field layout for this register. Bits[63:32] contain the CLOS field for each logical processor.

Note that placing the RMID field within the same PQR register enables both RMID and CLOS to be swapped at context swap time for simultaneous use of monitoring and allocation features with a single register write for efficiency.

When CDP is enabled, Specifying a CLOS value in IA32_PQR_ASSOC.CLOS greater than MAX_CLOS_CDP = (CPUID.10H.01H:EDX[15:0] >> 1) will cause undefined performance impact to code and data fetches. In all cases, code and data masks for L2 and L3 CDP should be programmed with at least one bit set.

Note that if the IA32_PQR_ASSOC.CLOS is never written then the CAT capability defaults to using CLOS 0, which in turn is set to the default mask in IA32_L3_MASK_0 - which is all “1”s (on reset). This essentially disables the enforcement feature by default or for legacy operating systems and software.

See Section 20.19.7, “Introduction to Memory Bandwidth Allocation,” for important CLOS programming considerations including maximum values when using CAT and CDP.

20.19.5 Code and Data Prioritization (CDP): Enumerating and Enabling L3 CDP Technology

L3 CDP is an extension of L3 CAT. The presence of the L3 CDP feature is enumerated via CPUID.10H.01H:ECX.CDP[2] (see Figure 20-32). Most of the CPUID.10H.01H sub-leaf data that applies to CAT also apply to CDP. However, CPUID.10H.01H:EDX.CAT_MAX_CLOS specifies the maximum CLOS applicable to CAT-only operation. For CDP operations, CLOS_MAX_CDP is equal to (CPUID.10H.01H:EDX.CAT_MAX_CLOS >> 1).

If CPUID.10H.01H:ECX.CDP[2] = 1, the processor supports CDP and provides a new MSR IA32_L3_QOS_CFG at address 0C81H. The layout of IA32_L3_QOS_CFG is shown in Figure 20-36. The bit field definition of IA32_L3_QOS_CFG are:

- Bit 0: L3 CDP Enable. If set, enables CDP, maps CAT mask MSRs into pairs of Data Mask and Code Mask MSRs. The maximum allowed value to write into IA32_PQR_ASSOC.CLOS is CLOS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

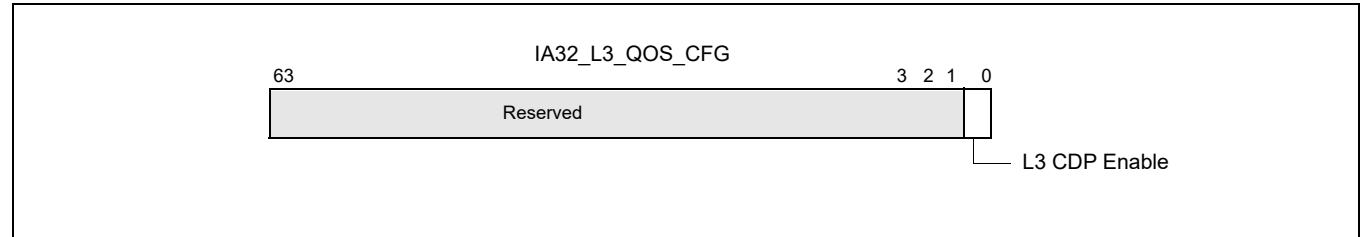


Figure 20-36. Layout of IA32_L3_QOS_CFG

IA32_L3_QOS_CFG default values are all 0s at RESET, the mask MSRs are all 1s. Hence, all logical processors are initialized in CLOS0 allocated with the entire L3 with CDP disabled, until software programs CAT and CDP. The scope of the IA32_L3_QOS_CFG MSR is defined to be the same scope as the L3 cache (e.g., typically per processor socket). Refer to Section 20.19.7 for software considerations while enabling or disabling L3 CDP.

20.19.5.1 Mapping Between L3 CDP Masks and CAT Masks

When CDP is enabled, the existing CAT mask MSR space is re-mapped to provide a code mask and a data mask per CLOS. The re-mapping is shown in Table 20-19.

Table 20-19. Re-indexing of CLOS Numbers and Mapping to CAT/CDP Mask MSRs

Mask MSR	CAT-only Operation	CDP Operation
IA32_L3_QOS_Mask_0	CLOS0	CLOS0.Data
IA32_L3_QOS_Mask_1	CLOS1	CLOS0.Code
IA32_L3_QOS_Mask_2	CLOS2	CLOS1.Data
IA32_L3_QOS_Mask_3	CLOS3	CLOS1.Code
IA32_L3_QOS_Mask_4	CLOS4	CLOS2.Data
IA32_L3_QOS_Mask_5	CLOS5	CLOS2.Code
....		
IA32_L3_QOS_Mask_‘2n’	CLOS‘2n’	CLOS‘n’.Data
IA32_L3_QOS_Mask_‘2n+1’	CLOS‘2n+1’	CLOS‘n’.Code

One can derive the MSR address for the data mask or code mask for a given CLOS number ‘n’ by:

- $\text{data\_mask\_address}(n) = \text{base} + (n \ll 1)$, where base is the address of IA32_L3_QOS_MASK_0.
- $\text{code\_mask\_address}(n) = \text{base} + (n \ll 1) + 1$.

When CDP is enabled, each CLOS is mapped 1:2 with mask MSRs, with one mask enabling programmatic control over data fill location and one mask enabling control over code placement. A variety of overlapped and isolated mask configurations are possible (see the example in Figure 20-29).

Mask MSR field definitions remain the same. Capacity masks must be formed of contiguous set bits, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type with a length of 1 bit or longer and should not exceed the maximum mask length specified in CPUID. As examples, valid masks on a cache with max bitmask length of 16b (from CPUID) include 0xFFFF, 0xFF00, 0x00FF, 0x00F0, 0x0001, 0x0003, and so on. Maximum valid mask lengths are unchanged whether CDP is enabled or disabled, and writes of invalid mask values may lead to undefined behavior. Writes to reserved bits will generate #GP(0).

20.19.6 Code and Data Prioritization (CDP): Enumerating and Enabling L2 CDP Technology

L2 CDP is an extension of the L2 CAT feature. The presence of the L2 CDP feature is enumerated via CPUID.10H.02H:ECX.CDP[2] (see Figure 17-33). Most of the CPUID.10H.02H sub-leaf data that applies to CAT also apply to CDP. However, CPUID.10H.02H:EDX.CAT_MAX_CLOS specifies the maximum CLOS applicable to CAT-only operation. For CDP operations, CLOS_MAX_CDP is equal to (CPUID.10H.02H:EDX.CAT_MAX_CLOS >> 1).

If CPUID.10H.02H:ECX.CDP[2] = 1, the processor supports L2 CDP and provides a new MSR IA32_L2_QOS_CFG at address 0C82H. The layout of IA32_L2_QOS_CFG is shown in Figure 20-37. The bit field definition of IA32_L2_QOS_CFG are:

- Bit 0: L2 CDP Enable. If set, enables CDP, maps CAT mask MSRs into pairs of Data Mask and Code Mask MSRs. The maximum allowed value to write into IA32_PQR_ASSOC.CLOS is CLOS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

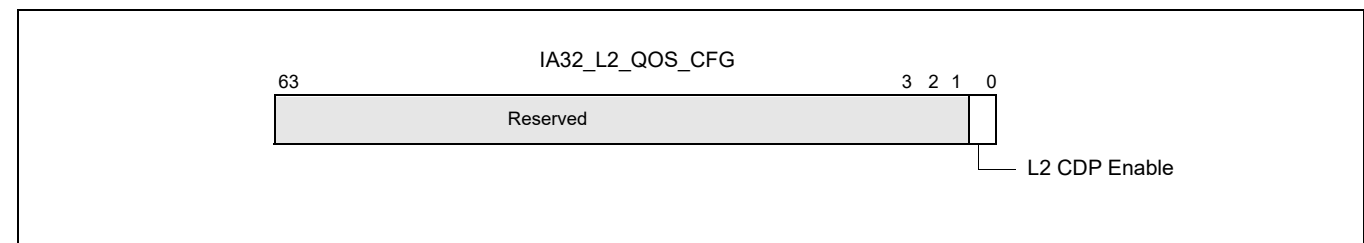


Figure 20-37. Layout of IA32_L2_QOS_CFG

IA32_L2_QOS_CFG default values are all 0s at RESET, and the mask MSRs are all 1s. Hence all logical processors are initialized in CLOS0 allocated with the entire L2 available and with CDP disabled, until software programs CAT and CDP. The IA32_L2_QOS_CFG MSR is defined at the same scope as the L2 cache, typically at the module level for Intel Atom processors for instance. In processors with multiple modules present it is recommended to program the IA32_L2_QOS_CFG MSR consistently across all modules for simplicity.

20.19.6.1 Mapping Between L2 CDP Masks and L2 CAT Masks

When CDP is enabled, the existing CAT mask MSR space is re-mapped to provide a code mask and a data mask per CLOS. This remapping is the same as the remapping shown in Table 20-19 for L3 CDP, but for the L2 MSR block (IA32_L2_QOS_MASK_n) instead of the L3 MSR block (IA32_L3_QOS_MASK_n). The same code / data mask mapping algorithm applies to remapping the MSR block between code and data masks.

As with L3 CDP, when L2 CDP is enabled, each CLOS is mapped 1:2 with mask MSRs, with one mask enabling programmatic control over data fill location and one mask enabling control over code placement. A variety of overlapped and isolated mask configurations are possible (see the example in Figure 20-29).

Mask MSR field definitions for L2 CDP remain the same as for L2 CAT. Capacity masks must be formed of contiguous set bits, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type with a length of 1 bit or longer and should not exceed the maximum mask length specified in CPUID. As examples, valid masks on a cache with max bitmask length of 16b (from CPUID) include 0xFFFF, 0xFF00, 0x00FF, 0x00F0, 0x0001, 0x0003, and so on. Maximum valid mask lengths are unchanged whether CDP is enabled or disabled, and writes of invalid mask values may lead to undefined behavior. Writes to reserved bits will generate #GP(0).

20.19.6.2 Common L2 and L3 CDP Programming Considerations

Before enabling or disabling L2 or L3 CDP, software should write all 1's to all of the corresponding CAT/CDP masks to ensure proper behavior (e.g., the IA32_L3_QOS_Mask_n set of MSRs for the L3 CAT feature). When enabling CDP, software should also ensure that only CLOS number which are valid in CDP operation is used, otherwise undefined behavior may result. For instance in a case with 16 CAT CLOS, since CLOS are reduced by half when CDP is enabled, software should ensure that only CLOS 0-7 are in use before enabling CDP (along with writing 1's to all mask bits before enabling or disabling CDP).

Software should also account for the fact that mask interpretations change when CDP is enabled or disabled, meaning for instance that a CAT mask for a given CLOS may become a code mask for a different Class of Service when CDP is enabled. In order to simplify this behavior and prevent unintended remapping software should consider resetting all threads to CLOS[0] before enabling or disabling CDP.

20.19.6.3 Cache Allocation Technology Dynamic Configuration

All Intel Resource Director Technology (Intel RDT) interfaces including the IA32_PQR_ASSOC MSR, CAT/CDP masks, MBA delay values, and CQM/MBM registers are accessible and modifiable at any time during execution using RDMSR/WRMSR unless otherwise noted. When writing to these MSRs a #GP(0) will be generated if any of the following conditions occur:

- A reserved bit is modified,

- Accessing a QOS mask register outside the supported CLOS (the max CLOS number is specified in CPUID.10H.ResID:EDX[15:0]), or
- Writing a CLOS greater than the supported maximum (specified as the maximum value of CPUID.10H.ResID:EDX[15:0] for all valid ResID values) is written to the IA32_PQR_ASSOC.CLOS field.

When CDP is enabled, specifying a CLOS value in IA32_PQR_ASSOC.CLOS outside of the lower half of the CLOS space will cause undefined performance impact to code and data fetches due to MSR space re-indexing into code/data masks when CDP is enabled.

When reading the IA32_PQR_ASSOC register the currently programmed CLOS on the core will be returned.

When reading an IA32_resourceType_MASK_n register the current capacity bit mask for CLOS 'n' will be returned.

As noted previously, software should minimize migrations of CLOS across logical processors (across threads or cores), as a reduction in the accuracy of the Cache Allocation feature may result if CLOS are migrated frequently. This is aligned with the industry standard practice of minimizing unnecessary thread migrations across processor cores in order to avoid excessive time spent warming up processor caches after a migration. In general, for best performance, minimize thread migration and CLOS migration across processor logical threads and processor cores.

20.19.6.4 Cache Allocation Technology Operation With Power Saving Features

Note that the Cache Allocation Technology feature cannot be used to enforce cache coherency, and that some advanced power management features such as C-states which may shrink or power off various caches within the system may interfere with CAT hints - in such cases the CAT bitmasks are ignored and the other features take precedence. If the highest possible level of CAT differentiation or determinism is required, disable any power-saving features which shrink the caches or power off caches. The details of the power management interfaces are typically implementation-specific, but can be found at Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

If software requires differentiation between threads but not absolute determinism then in many cases it is possible to leave power-saving cache shrink features enabled, which can provide substantial power savings and increase battery life in mobile platforms. In such cases when the caches are powered off (e.g., package C-states) the entire cache of a portion thereof may be powered off. Upon resuming an active state any new incoming data to the cache will be filled subject to the cache capacity bitmasks. Any data in the cache prior to the cache shrink or power off may have been flushed to memory during the process of entering the idle state, however, and is not guaranteed to remain in the cache. If differentiation between threads is the goal of system software then this model allows substantial power savings while continuing to deliver performance differentiation. If system software needs optimal determinism then power saving modes which flush portions of the caches and power them off should be disabled.

NOTE

IA32_PQR_ASSOC is saved and restored across C6 entry/exit. Similarly, the mask register contents are saved across package C-state entry/exit and are not lost.

20.19.6.5 Cache Allocation Technology Operation with Other Operating Modes

The states in IA32_PQR_ASSOC and mask registers are unmodified across an SMI delivery. Thus, the execution of SMM handler code can interact with the Cache Allocation Technology resource and manifest some degree of non-determinism to the non-SMM software stack. An SMM handler may also perform certain system-level or power management practices that affect CAT operation.

It is possible for an SMM handler to minimize the impact on data determinism in the cache by reserving a CLOS with a dedicated partition in the cache. Such an SMM handler can switch to the dedicated CLOS immediately upon entering SMM, and switching back to the previously running CLOS upon exit.

20.19.6.6 Associating Threads with CAT/CDP Classes of Service

Threads are associated with Classes of Service (CLOS) via the per-logical-processor IA32_PQR_ASSOC MSR. The same CLOS concept applies to both CAT and CDP (for instance, CLOS[5] means the same thing whether CAT or CDP

is in use, and the CLOS has associated resource usage constraint attributes including cache capacity masks). The mapping of CLOS to mask MSRs does change when CDP is enabled, according to the following guidelines:

- In CAT-only Mode - one set of bitmasks in one mask MSR control both code and data.
 - Each CLOS number map 1:1 with a capacity mask on the applicable resource (e.g., L3 cache).
- When CDP is enabled:
 - Two mask sets exist for each CLOS number, one for code, one for data.
 - Masks for code/data are interleaved in the MSR address space (see Table 20-19).

20.19.7 Introduction to Memory Bandwidth Allocation

The Memory Bandwidth Allocation (MBA) feature provides indirect and approximate control over memory bandwidth available per-core. It was introduced in the Intel Xeon Scalable Processor Family. This feature provides a method to control applications that may be over-utilizing bandwidth relative to their priority in environments such as the data-center.

The MBA feature uses existing constructs from the Intel RDT feature set, including Classes of Service (CLOS). A given CLOS used for L3 CAT, for instance, means the same thing as a CLOS used for MBA. Infrastructure, such as the MSR used to associate a thread with a CLOS (the IA32_PQR_ASSOC_MSR) and some elements of the CPUID enumeration (such as CPUID.10H), are shared. Certain generations include advanced hardware controllers for efficiency. For more information, refer to the “Intel® Resource Director Technology Architecture Specification.”

The following sections describe CPU interfaces to Memory Bandwidth Allocation, such as CPUID enumeration and configuration interfaces (MSRs).

20.19.7.1 Memory Bandwidth Allocation Enumeration

Similar to other Intel RDT features, enumeration of the presence and details of the MBA feature is provided via a sub-leaf of the CPUID instruction.

Key components of the enumeration are as follows.

- Support for the MBA feature on the processor, and if MBA is supported, the following details:
 - Number of supported Classes of Service (CLOS) for the processor.
 - The maximum MBA delay value supported (which also implicitly provides a definition of the granularity).
 - An indication of whether the delay values which can be programmed are linearly spaced or not.

The presence of any of the Intel RDT features which enable control over shared platform resources is enumerated by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.07H.00H:EBX.RDT_A[15] reports 1, the processor supports software control over shared processor resources. Software may then use CPUID.10H to enumerate additional details on the specific controls provided.

Through CPUID.10H software may determine whether MBA is supported on the platform. Specifically, as shown in Figure 20-31, bit 3 of the EBX register indicates whether MBA is supported on the processor, and the bit position (3) constitutes a Resource ID (ResID) which allows enumeration of MBA details. For instance, if bit 3 is supported this implies the presence of CPUID.10H.[ResID=3] as shown in Figure 20-38 which provides the following details.

- CPUID.10H.ResID=3:EAX[11:0] reports the maximum MBA throttling value supported, minus one. For instance, a value of 89 indicates that a maximum throttling value of 90 is supported. Additionally, in cases where a linear interface (see below) is supported then one hundred minus the maximum throttling value indicates the granularity, 10% in this example.
- CPUID.10H.ResID=3:EBX is reserved.
- CPUID.10H.ResID=3:ECX[0] indicates that the MBA throttling values are applied per-logical-processor independently if set.
- CPUID.10H.ResID=3:ECX[2] reports whether the response of the delay values is linear (see text).
- CPUID.10H.ResID=3:EDX[15:0] reports the number of Classes of Service (CLOS) supported for the feature (minus one). For instance, a reported value of 15 implies a maximum of 16 supported MBA CLOS.

The number of CLOS supported for the MBA feature may or may not align with other resources such as L3 CAT. In cases where the Intel RDT features support different numbers of CLOS the lowest numerical CLOS support the common set of features, while higher CLOS may support a subset. For instance, if L3 CAT supports 8 CLOS while MBA supports 4 CLOS, all 8 CLOS would have L3 CAT masks available for cache control, but the upper 4 CLOS would not offer MBA support. In this case the upper 4 CLOS would not be subject to any throttling control. Software can manage supported resources / CLOS in order to either have consistent capabilities across CLOS by using the common subset or enable more flexibility by selectively applying resource control where needed based on careful CLOS and thread mapping. In all cases, CLOS[0] supports all Intel RDT resource control features present on the platform.

Discussion on the interpretation and usage of the MBA delay values is provided in Section 20.19.7.2 on MBA configuration.

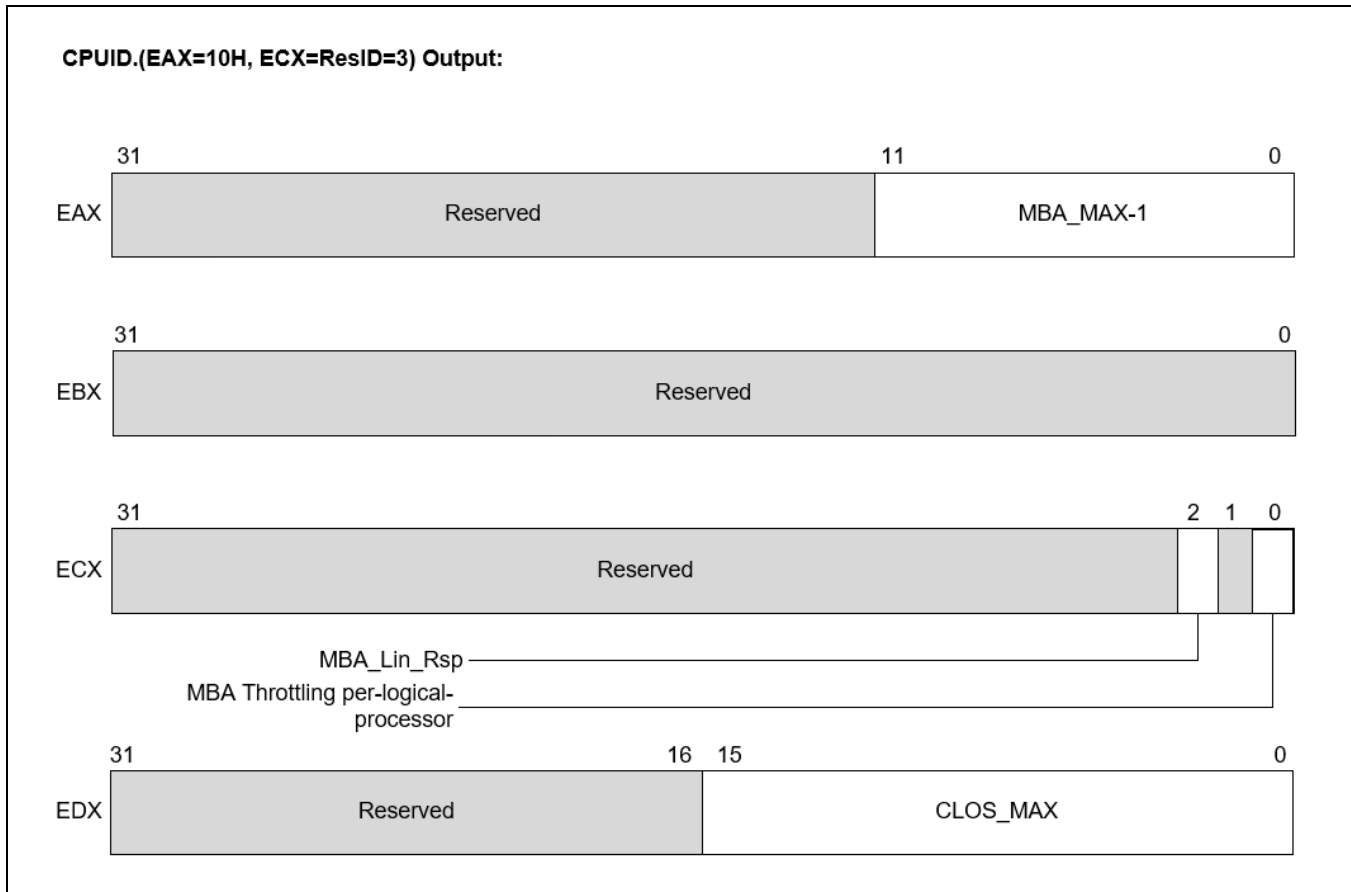


Figure 20-38. CPUID.10H.03H MBA Feature Details Identification

20.19.7.2 Memory Bandwidth Allocation Configuration

The configuration of MBA takes consists of two processes once enumeration is complete.

- Association of threads to Classes of Service (CLOS) - accomplished in a common fashion across Intel RDT features as described in Section 20.19.7.1 via the `IA32_PQR_ASSOC` MSR. As with features such as L3 CAT, software may update the CLOS field of the PQR MSR at context swap time in order to maintain the proper association of software threads to Classes of Service on the hardware. While logical processors may each be associated with independent CLOS, see Section 20.19.7.3 for important usage model considerations (initial versions of the MBA feature select the maximum delay value across threads).
- Configuration of the per-CLOS delay values, accomplished via the `IA32_L2_QoS_Ext_BW_Thrtl_n` MSR set shown in Table 20-20.

The MBA delay values which may be programmed range from zero (implying zero delay, and full bandwidth available) to the maximum (MBA_MAX) specified in CPUID as discussed in Section 20.19.7.1. The throttling values are approximate and do not sum to 100% across CLOS, rather they should be viewed as a maximum bandwidth “cap” per-CLOS.

Software may select an MBA delay value then write the value into one or more of the IA32_L2_QoS_Ext_BW_Thrtl_n MSRs to update the delay values applied for a specific CLOS. As shown in Table 20-20 the base address of the MSRs is at D50H, and the range corresponds to the maximum supported CLOS from CPUID.10H.ResID=1:EDX[15:0] as described in Section 20.19.7.1. For instance, if 16 CLOS are supported then the valid MSR range will extend from D50H through D5F inclusive.

Table 20-20. MBA Delay Value MSRs

Delay Value MSR	Address
IA32_L2_QoS_Ext_BW_Thrtl_0	D50H
IA32_L2_QoS_Ext_BW_Thrtl_1	D51H
IA32_L2_QoS_Ext_BW_Thrtl_2	D52H
....	
IA32_L2_QoS_Ext_BW_Thrtl_‘CLOS_MAX’	D50H + CLOS_MAX from CPUID.10H.03H

The definition for the MBA delay value MSRs is provided in Figure 17.39. The lower 16 bits are used for MBA delay values, and values from zero to the maximum from the CPUID MBA_MAX-1 value are supported. Values outside this range will generate #GP(0).

If linear input throttling values are indicated by CPUID.10H.ResID=3:ECX[2] then values from zero through the MBA_MAX field from CPUID.10H.ResID=3:EAX[11:0] are supported as inputs. In the linear mode the input precision is defined as 100-(MBA_MAX). For instance, if the MBA_MAX value is 90, the input precision is 10%. Values not an even multiple of the precision (e.g., 12%) will be rounded down (e.g., to 10% delay applied).

- If linear values are not supported (CPUID.10H.ResID=3:ECX[2] = 0) then input delay values are powers-of-two from zero to the MBA_MAX value from CPUID. In this case any values not a power of two will be rounded down the next nearest power of two.

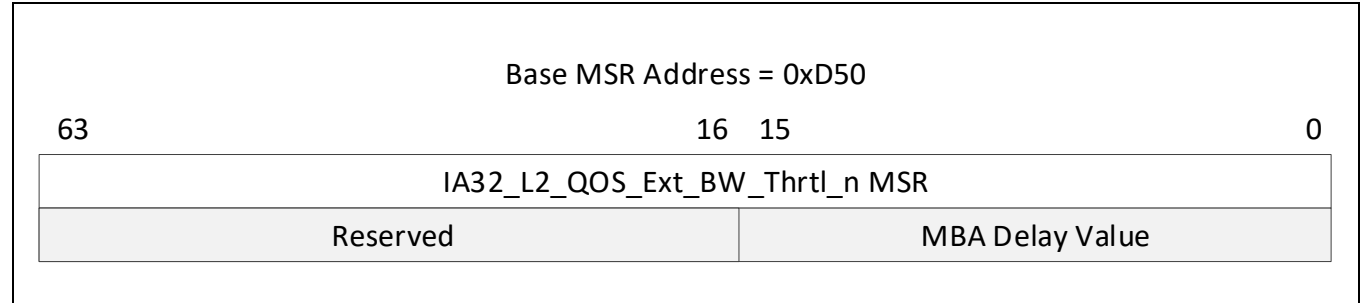


Figure 20-39. IA32_L2_QoS_Ext_BW_Thrtl_n MSR Definition

Note that the throttling values provided to software are calibrated through specific traffic patterns, however as workload characteristics may vary the response precision and linearity of the delay values will vary across products and should be treated as approximate values only.

20.19.7.3 Memory Bandwidth Allocation Usage Considerations

Different versions of Memory Bandwidth Allocation have various usage considerations and improving efficiency over time. See the “Intel® Resource Director Technology Architecture Specification” for additional details.

20.20 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) FOR NON-CPU AGENTS

This section describes Intel RDT features for non-CPU agents. CPU agents are threads running on IA cores. Non-CPU agents include PCIe and CXL devices and integrated accelerators, thus broadly encompassing the set of agents that read from and write to either caches or memory, excluding IA cores. The non-CPU agent Intel RDT features enable monitoring of I/O device shared cache and memory bandwidth and cache allocation control. This provides features for I/O devices equivalent to the CPU agent Intel RDT capabilities CMT, MBM, and CAT (discussed in Section 20.18 and Section 20.19). Refer to the “Intel® Resource Director Technology Architecture Specification” regarding design goals, use cases, software architecture, ACPI enumeration, and MMIO register interfaces.

“Non-CPU agent Intel RDT” refers to capabilities that monitor and control non-CPU agents' resource utilization, including PCIe and CXL devices and integrated accelerators. Non-CPU agent Intel RDT may be called I/O RDT in some literature. In this document, the term “non-CPU agent Intel RDT” is used.

20.20.1 Non-CPU Agent Intel® RDT Features Enumeration Details

CPU agent Intel RDT features use the CPUID instruction to enumerate supported features and the level of support. Architectural Model-Specific Registers (MSRs) are interfaces to the monitoring and allocation features, as described in Sections 18.18 and 18.19.

Non-CPU agent Intel RDT builds on CPU agent Intel RDT by extending CPUID to indicate the presence and integration of non-CPU agent Intel RDT and by providing rich enumeration information in vendor-specific extensions to the Advanced Configuration and Power Interface (ACPI), in particular in the I/O RDT (IRDT) table. The ACPI extensions detailed in the “Intel® Resource Director Technology Architecture Specification” provide mechanisms to comprehend the structure of devices attached behind I/O blocks to particular links and what forms of tagging are supported on a per-link basis.

It is recommended that software parse CPUID and ACPI to obtain a detailed understanding of platform support and capabilities before attempting to use non-CPU agent Intel RDT.

20.20.1.1 CPUID-Based Enumeration for Non-CPU Agent Intel® RDT Feature

CPUID-based enumeration provides a method by which all architectural Intel RDT features may be enumerated.

For CPU agent Intel RDT, monitoring details are enumerated in a CPUID sub-leaf denoted as CPUID.0FH.ResID, where ResID corresponds to a resource ID bit index from the CPUID.0FH.00H sub-leaf. Similarly, Intel RDT allocation features are described in CPUID.10H.ResID. (Note that the ResID bit positions are not guaranteed to be symmetric or have the same encodings.)

No CPUID leaves or sub-leaves are created for non-CPU agent Intel RDT. Rather, non-CPU agent Intel RDT extends the existing Intel RDT CPUID sub-leaves with a bit per resource type, indicating whether non-CPU agent Intel RDT monitoring or control is present. CPUID.0FH.ResID=1:EAX[10:9] represents the presence of CMT and MBM features for non-CPU agents. CPUID.10H.ResID=1:ECX[1] represents the presence of the CAT feature for non-CPU agents.

Specifically, for non-CPU Agent Intel RDT Monitoring (see Figure 20-21):

- Bits are added in the CPU Agent Intel RDT CMT/MBM leaf: CPUID.0FH.ResID=1:EAX[10:9].
 - EAX[bit 9]: If set, indicates the presence of non-CPU Agent Cache Occupancy Monitoring (the equivalent of CPU Agent Intel RDT's CMT feature).
 - EAX[bit 10]: If set, indicates the presence of non-CPU Agent memory L3 external BW monitoring (the equivalent of CPU Agent Intel RDT's MBM feature).

For non-CPU Agent Intel RDT Allocation (see Figure 20-32):

- New bit in L3 CAT leaf: CPUID.10H.ResID=1:ECX[1].
 - ECX[bit 1]: If set, indicates the presence of non-CPU Agent Cache Allocation Technology (the equivalent of CPU Agent Intel RDT's L3 CAT feature).
- As before, ECX[bit 2] indicates that L3 CDP is supported if set.

Note that no equivalent bits are defined in CPUID.10H.ResID=2 as there is no ability for devices to fill into core L2 caches.

If any of these non-CPU agent Intel RDT enumeration bits are set, indicating that a monitoring feature or allocation feature is present, it also indicates the presence of the IA32_L3_IO_RDT_CFG architectural MSR. This MSR may be used to enable the non-CPU agent Intel RDT features. See Section 20.20.2 for MSR details.

The presence of Intel RDT is a prerequisite for using the equivalent non-CPU agent Intel RDT feature. If a particular CPU agent Intel RDT feature is absent, any attempt to use non-CPU agent Intel RDT equivalents will result in general protection faults in the MSR interface. Attempts to enable unsupported features in the I/O complex will result in writes to the corresponding MMIO enable or configuration interfaces being ignored.

Software may use the existing CPUID leaves to gather the maximum number of RMID and CLOS tags for each resource level (e.g., L3 cache), and non-CPU agent Intel RDT is also subject to these limits.

Some platforms may support a mix of features, for instance, supporting L3 CAT architectural controls and the non-CPU agent Intel RDT equivalent, but no CMT/MBM monitoring or non-CPU agent monitoring equivalent, and these capabilities should be enumerated on a per-platform basis.

20.20.1.2 ACPI Enumeration

When support for non-CPU agent Intel RDT features is detected using CPUID, ACPI may be consulted for further details on the level of feature support, device structures behind various I/O ports, and the specific MMIO interfaces used to control a given device.

Non-CPU agent Intel RDT enumeration is via the “IRDT” ACPI table. For more information, refer to the “Intel® Resource Director Technology Architecture Specification.”

20.20.2 Non-CPU Agent Intel® RDT Feature Enable MSR

Before configuring non-CPU agent Intel RDT through MMIO, the feature should be enabled using the non-CPU agent Intel RDT Feature Enable MSR, IA32_L3_IO_RDT_CFG (MSR address 0C83H). As described in Section 20.20.1.1, the presence of one or more CPUID bits indicating support for one or more non-CPU agent Intel RDT features also indicates the presence of this MSR. This MSR may be used to enable the non-CPU agent Intel RDT features.

Two bits are defined in this MSR. Bit 0, when set, enables non-CPU agent RDT resource allocation features. Bit 1, when set, enables non-CPU agent Intel RDT monitoring features.

The L3 Non-CPU agent Intel RDT Monitoring Enable bit is supported if CPUID indicates that one or more non-CPU agent Intel RDT resource monitoring features are present.

The L3 Non-CPU agent Intel RDT Allocation Enable bit is supported if CPUID indicates that one or more non-CPU agent Intel RDT resource allocation features are present.

The default value is 0x0, so both classes of features are disabled by default. All bits not defined are reserved. Writing a non-zero value to any reserved bit will generate a General Protection Fault (#GP(0)).

This MSR is scoped at the L3 cache level and is cleared on system reset. It is expected that the software will configure this MSR consistently across all L3 caches that may be present on that package.

The definition of the IA32_L3_IO_RDT_CFG MSR is shown in Figure 20-40.

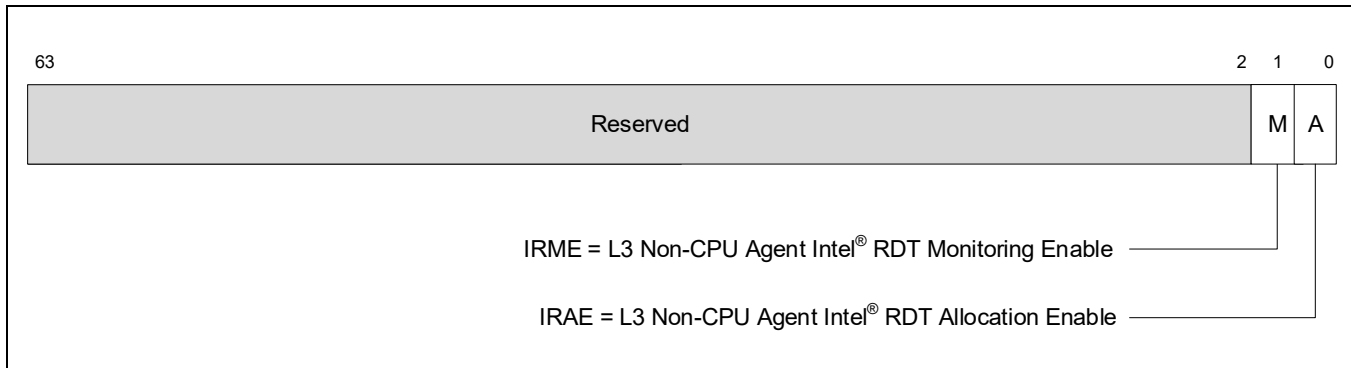


Figure 20-40. Layout of the IA32_L3_IO_RDT_CFG MSR for Enabling Non-CPU Agent Intel® RDT

20.20.3 Asymmetrical Enumeration of Intel® RDT Features

The enumeration of RDT has been extended to platforms that enumerate asymmetrical resources. For example, hybrid platforms (supporting multiple processor types) may contain multiple last-level caches (LLCs) with different capabilities, or some processors may not include an LLC at the same enumeration level. Software should consider the following changes to RDT enumeration to enable full RDT capabilities across all platforms. Asymmetrical RDT CPUID leaves 27H and 28H may enumerate distinct capabilities between logical processors on the platform, while CPUID RDT leaves 0FH and 10H may enumerate the common set of capabilities to ensure software compatibility. Asymmetrical RDT CPUID leaves should be enumerated per logical processor to identify differences in capabilities.

The enumeration of RDT technologies first consists of the following four CPUID leaf 07H bits in Table 20-21 and the corresponding CPUID leaves in Table 20-22.

Table 20-21. Asymmetrical Enumeration

CPUID Enumeration	Field Name	Description
CPUID.(EAX=07H,ECX=00H):EBX. RDT-M[bit 12]	RDT-M	Specifies that the RDT-M symmetrical enumeration is available on all logical processors.
CPUID.(EAX=07H,ECX=00H):EBX. RDT-A[bit 15]	RDT-A	Specifies that the RDT-A symmetrical enumeration is available on all logical processors.
CPUID.(EAX=07H,ECX=00H):ECX. Asymmetric RDT-M[bit 0]	Asymmetric RDT-M	Specifies that logical processors on this platform may support RDT-M; but is not guaranteed.
CPUID.(EAX=07H,ECX=00H):ECX. Asymmetric RDT-A[bit 1]	Asymmetric RDT-A	Specifies that logical processors on this platform may support RDT-A; but is not guaranteed.

Table 20-22. Asymmetrical CPUID Leaves

CPUID Enumeration	CPUID Leaf	Description
CPUID.(EAX=07H,ECX=00H):EBX. X.RDT-M[bit 12]	Leaf 0FH	This leaf provides the same values regardless of the logical processor querying leaf 0FH.
CPUID.(EAX=07H,ECX=00H):EBX. X.RDT-A[15]	Leaf 10H	This leaf provides the same values regardless of the logical processor querying leaf 10H.
CPUID.(EAX=07H,ECX=00H):ECX. X.Asymmetric RDT-M[bit 0]	Leaf 27H	This leaf can return different values between logical processors querying leaf 27H.
CPUID.(EAX=07H,ECX=00H):ECX. X.Asymmetric RDT-A[bit 1]	Leaf 28H	This leaf can return different values between logical processors querying leaf 28H.

The Asymmetrical RDT-M leaf 27H and subleaves have the same definitions as RDT-M leaf 0FH. The Asymmetrical RDT-A leaf 28H and subleaves have the same definitions as RDT-A leaf 10H. Symmetrical platforms may enumerate the existence of asymmetrical leaves, and therefore cannot be used to determine a system is asymmetrical.

The data provided in the asymmetrical leaves 27H and 28H are supersets of leaves 0FH and 10H. The enumeration and capabilities returned through these leaves can be any of the following:

- Enumerating either leaves 27H or 28H is not indicative that leaves 0FH or 10H would be supported.
- Leaves 0FH and/or 10H may return a smaller subset than the intersection of features across resources where either leaves 27H or 28H exist on a platform.
- Leaves 0FH and/or 10H may return the intersection of features across resources where either leaves 27H or 28H exist on a platform.
- Leaves 0FH and/or 10H may be enumerated but contain no information where either leaves 27H or 28H exist on a platform.

The enumeration provided in leaves 0FH and 10H do not have a requirement to provide resource feature information on a platform where leaves 27H or 28H exist.

Software should use the following algorithm to detect support of Intel® RDT Monitoring:

```
IF CPUID.(EAX=07H,ECX=01H):ECX.Asymmetrical RDT-M[Bit 0] = 1 THEN
    // Perform asymmetrical enumeration using leaf 27H
ELSE IF CPUID.(EAX=07H,ECX=01H):EBX.RDT-M[Bit 12] = 1 THEN
    // Perform symmetrical enumeration using leaf 0FH
ELSE
    // Intel® Resource Directory Technology Monitoring is not supported.
ENDIF
ENDIF
```

Software should use the algorithm below to detect support of Intel® RDT Allocation:

```
IF CPUID.(EAX=07H,ECX=01H):ECX.Asymmetrical RDT-A[Bit 1] = 1 THEN
    // Perform asymmetrical enumeration using leaf 28H
ELSE IF CPUID.(EAX=07H,ECX=01H):EBX.RDT-A[Bit 15] = 1 THEN
    // Perform symmetrical enumeration using leaf 10H
ELSE
    // Intel® Resource Directory Technology Allocation is not supported.
ENDIF
ENDIF
```

The asymmetrical leaves are required to be read on each logical processor. Software may determine the intersection of features across all logical processors, or software may group logical processors with matching feature sets. Logical processors that share a resource will report the same feature support for that resource. An example would be to enumerate each set of logical processors sharing an L3 cache, reading CPUID leaf 28H for L3 CAT on one logical processor sharing each L3 cache and applying those settings to all logical processors sharing the same L3 cache.

20.21 CORE TELEMETRY FOR INTEL® PLATFORM MONITORING TECHNOLOGY (PMT)

Core telemetry is part of the Intel® Platform Monitoring Technology (Intel® PMT) feature set. Processors that support core telemetry collect per-core telemetry during core execution (including the execution context of the core — e.g., resource monitoring ID or RMID), aggregate the data internally, and report them to external agents, enabling workload-level tracking. This telemetry can be used by the external agents to monitor processor behavior to improve placement of workloads in data centers, increase resource utilization, monitor energy consumption, etc. The telemetry collected is processor-specific. For information about Intel® PMT refer to Intel® Platform Monitoring Technology (Intel® PMT) External Specification.

To avoid creating a side channel that might leak information from trusted execution environments (TEEs), core telemetry collected during TEE execution is discarded by the processor. Specifically, out-of-band core telemetry is discarded if it was collected while the core was running in any of the following situations:

- During execution of an enclave supported by Intel® Software Guard Extensions (Intel® SGX). This is true even for enclave debug following an opt-in entry.
- During execution of an authenticated code module.
- During execution in SEAM root operation.
- During execution in SEAM non-root operation, unless the “allow SEAM-guest telemetry” VM-entry control is 1.

For information about SEAM (secure arbitration mode), see Chapter 35.

20. Updates to Chapter 22, Volume 3B

Change bars and text show changes to Chapter 22 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- In Figure 22-14, "IA32_PMC_GPx_CFG_A MSR (also known as IA32_PERFVTSELx) Supporting Architectural Performance Monitoring Version 6" (in Section 22.2.6.1), removed the Any Thread bit (bit 21) as it is not supported in Version 6
- Added new Section 22.10, "Architectural PEBS" and re-numbered subsequent sections.

Intel 64 and IA-32 architectures provide facilities for monitoring performance via a PMU (Performance Monitoring Unit).

NOTE

Performance monitoring events can be found here: <https://perfmon-events.intel.com/>.

Additionally, performance monitoring event files for Intel processors are maintained by Intel. These files can be downloaded here: <https://github.com/intel/perfmon>.

22.1 PERFORMANCE MONITORING OVERVIEW

Performance monitoring was introduced in the Pentium processor with a set of model-specific performance-monitoring counter MSRs. These counters permit selection of processor performance parameters to be monitored and measured. The information obtained from these counters can be used for tuning system and compiler performance.

In Intel P6 family of processors, the performance monitoring mechanism was enhanced to permit a wider selection of events to be monitored and to allow greater control events to be monitored. Next, Intel processors based on Intel NetBurst microarchitecture introduced a distributed style of performance monitoring mechanism and performance events.

The performance monitoring mechanisms and performance events defined for the Pentium, P6 family, and Intel processors based on Intel NetBurst microarchitecture are not architectural. They are all model specific (not compatible among processor families). Intel Core Solo and Intel Core Duo processors support a set of architectural performance events and a set of non-architectural performance events. Newer Intel processor generations support enhanced architectural performance events and non-architectural performance events.

Starting with Intel Core Solo and Intel Core Duo processors, there are two classes of performance monitoring capabilities. The first class supports events for monitoring performance using counting or interrupt-based event sampling usage. These events are non-architectural and vary from one processor model to another. They are similar to those available in Pentium M processors. These non-architectural performance monitoring events are specific to the microarchitecture and may change with enhancements. They are discussed in Section 22.6.3, "Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)." Non-architectural events for a given microarchitecture cannot be enumerated using CPUID; and they can be found at: <https://perfmon-events.intel.com/> or at <https://github.com/intel/perfmon/>.

The second class of performance monitoring capabilities is referred to as architectural performance monitoring. This class supports the same counting and Interrupt-based event sampling usages, with a smaller set of available events. The visible behavior of architectural performance events is consistent across processor implementations. Availability of architectural performance monitoring capabilities is enumerated using the CPUID.0AH. These events are discussed in Section 22.2.

See also:

- Section 22.2, "Architectural Performance Monitoring."
- Section 22.3, "Performance Monitoring (Intel® Core™ Processors and Intel® Xeon® Processors)."
 - Section 22.3.1, "Performance Monitoring for Processors Based on Nehalem Microarchitecture."
 - Section 22.3.2, "Performance Monitoring for Processors Based on Westmere Microarchitecture."
 - Section 22.3.3, "Intel® Xeon® Processor E7 Family Performance Monitoring Facility."
 - Section 22.3.4, "Performance Monitoring for Processors Based on Sandy Bridge Microarchitecture."
 - Section 22.3.5, "3rd Generation Intel® Core™ Processor Performance Monitoring Facility."
 - Section 22.3.6, "4th Generation Intel® Core™ Processor Performance Monitoring Facility."

- Section 22.3.7, "5th Generation Intel® Core™ Processor and Intel® Core™ M Processor Performance Monitoring Facility."
- Section 22.3.8, "6th Generation, 7th Generation and 8th Generation Intel® Core™ Processor Performance Monitoring Facility."
- Section 22.3.9, "10th Generation Intel® Core™ Processor Performance Monitoring Facility."
- Section 22.3.10, "12th and 13th Generation Intel® Core™ Processors, and 4th and 5th Generation Intel® Xeon® Scalable Processor Family Performance Monitoring Facility."
- Section 22.3.11, "Intel® Series 2 Core™ Ultra Processor Performance Monitoring Facility."
- Section 22.4, "Performance monitoring (Intel® Xeon™ Phi Processors)."
- Section 22.4.1, "Intel® Xeon Phi™ Processor 7200/5200/3200 Performance Monitoring."
- Section 22.5, "Performance Monitoring (Intel Atom® Processors)."
- Section 22.5.1, "Performance Monitoring (45 nm and 32 nm Intel Atom® Processors)."
- Section 22.5.2, "Performance Monitoring for Silvermont Microarchitecture."
- Section 22.5.3, "Performance Monitoring for Goldmont Microarchitecture."
- Section 22.5.4, "Performance Monitoring for Goldmont Plus Microarchitecture."
- Section 22.5.5, "Performance Monitoring for Tremont Microarchitecture."
- Section 22.6, "Performance Monitoring (Legacy Intel Processors)."
- Section 22.6.1, "Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)."
- Section 22.6.2, "Performance Monitoring (Processors Based on Intel® Core™ Microarchitecture)."
- Section 22.6.3, "Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)."
- Section 22.6.4, "Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture."
- Section 22.6.4.5, "Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture."
- Section 22.6.5, "Performance Monitoring and Dual-Core Technology."
- Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."
- Section 22.6.7, "Performance Monitoring on L3 and Caching Bus Controller Sub-Systems."
- Section 22.6.8, "Performance Monitoring (P6 Family Processor)."
- Section 22.6.9, "Performance Monitoring (Pentium Processors)."
- Section 22.7, "Counting Clocks."
- Section 22.8, "IA32_PERF_CAPABILITIES MSR Enumeration."
- Section 22.9, "PEBS Facility."
- Section 22.10, "Architectural PEBS."

22.2 ARCHITECTURAL PERFORMANCE MONITORING

Performance monitoring events are architectural when they behave consistently across microarchitectures. Intel Core Solo and Intel Core Duo processors introduced architectural performance monitoring. The feature provides a mechanism for software to enumerate performance events and provides configuration and counting facilities for events.

Architectural performance monitoring does allow for enhancement across processor implementations. The CPUID.0AH leaf provides version ID for each enhancement. Intel Core Solo and Intel Core Duo processors support base level functionality identified by version ID of 1. Processors based on Intel Core microarchitecture support, at a minimum, the base level functionality of architectural performance monitoring. Intel Core 2 Duo processor T

7700 and newer processors based on Intel Core microarchitecture support both the base level functionality and enhanced architectural performance monitoring identified by version ID of 2.

45 nm and 32 nm Intel Atom processors and Intel Atom processors based on the Silvermont microarchitecture support the functionality provided by versionID 1, 2, and 3; CPUID.0AH:EAX[7:0] reports versionID = 3 to indicate the aggregate of architectural performance monitoring capabilities. Intel Atom processors based on the Airmont microarchitecture support the same performance monitoring capabilities as those based on the Silvermont microarchitecture. Intel Atom processors based on the Goldmont and Goldmont Plus microarchitectures support versionID 4. Intel Atom processors starting with processors based on the Tremont microarchitecture support versionID 5.

Intel Core processors and related Intel Xeon processor families based on the Nehalem through Broadwell microarchitectures support version ID 3. Intel processors based on the Skylake through Coffee Lake microarchitectures support versionID 4. Intel processors starting with processors based on the Ice Lake microarchitecture support versionID 5.

22.2.1 Architectural Performance Monitoring Version 1

Configuring an architectural performance monitoring event involves programming performance event select registers. There are a finite number of performance event select MSRs (IA32_PERFEVTSELx MSRs). The result of a performance monitoring event is reported in a performance monitoring counter (IA32_PMCx MSR). Performance monitoring counters are paired with performance monitoring select registers.

Performance monitoring select registers and counters are architectural in the following respects:

- The bit field layout of IA32_PERFEVTSELx is consistent across microarchitectures. A non-zero write of a field that is introduced after the initial implementation of architectural performance monitoring (Version 1) results in #GP if that field is not supported.
- Addresses of IA32_PERFEVTSELx MSRs remain the same across microarchitectures.
- Addresses of IA32_PMC MSRs remain the same across microarchitectures.
- Each logical processor has its own set of IA32_PERFEVTSELx and IA32_PMCx MSRs. Configuration facilities and counters are not shared between logical processors sharing a processor core.

Architectural performance monitoring provides a CPUID mechanism for enumerating the following information:

- Number of performance monitoring counters available to software in a logical processor (each IA32_PERFEVTSELx MSR is paired to the corresponding IA32_PMCx MSR).
- Number of bits supported in each IA32_PMCx.
- Number of architectural performance monitoring events supported in a logical processor.

Software can use CPUID to discover architectural performance monitoring availability (CPUID.0AH). The architectural performance monitoring leaf provides an identifier corresponding to the version number of architectural performance monitoring available in the processor.

The version identifier is retrieved by querying CPUID.0AH:EAX[7:0] (see Chapter 3, “Instruction Set Reference, A-L,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A). If the version identifier is greater than zero, architectural performance monitoring capability is supported. Software queries the CPUID.0AH for the version identifier first; it then analyzes the value returned in CPUID.0AH:EAX, CPUID.0AH:EBX to determine the facilities available.

In the initial implementation of architectural performance monitoring; software can determine how many IA32_PERFEVTSELx/ IA32_PMCx MSR pairs are supported per core, the bit-width of PMC, and the number of architectural performance monitoring events available.

Architectural performance monitoring facilities include a set of performance monitoring counters and performance event select registers. These MSRs have the following properties:

- IA32_PMCx MSRs start at address 0C1H and occupy a contiguous block of MSR address space; the number of MSRs per logical processor is reported using CPUID.0AH:EAX[15:8]. Note that this may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters.

- IA32_PERFEVTSELx MSRs start at address 186H and occupy a contiguous block of MSR address space. Each performance event select register is paired with a corresponding performance counter in the 0C1H address block. Note the number of IA32_PERFEVTSELx MSRs may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters.
- The bit width of an IA32_PMCx MSR is reported using the CPUID.0AH:EAX[23:16]. This the number of valid bits for read operation. On write operations, the lower-order 32 bits of the MSR may be written with any value, and the high-order bits are sign-extended from the value of bit 31.
- Bit field layout of IA32_PERFEVTSELx MSRs is defined architecturally.

See Figure 22-1 for the bit field layout of IA32_PERFEVTSELx MSRs. The bit fields are:

- **Event select field (bits 0 through 7)** — Selects the event logic unit used to detect microarchitectural conditions (see Table 22-3, for a list of architectural events and their 8-bit codes). The set of values for this field is defined architecturally; each value corresponds to an event logic unit for use with an architectural performance event. The number of architectural events is queried using CPUID.0AH:EAX. A processor may support only a subset of pre-defined values.

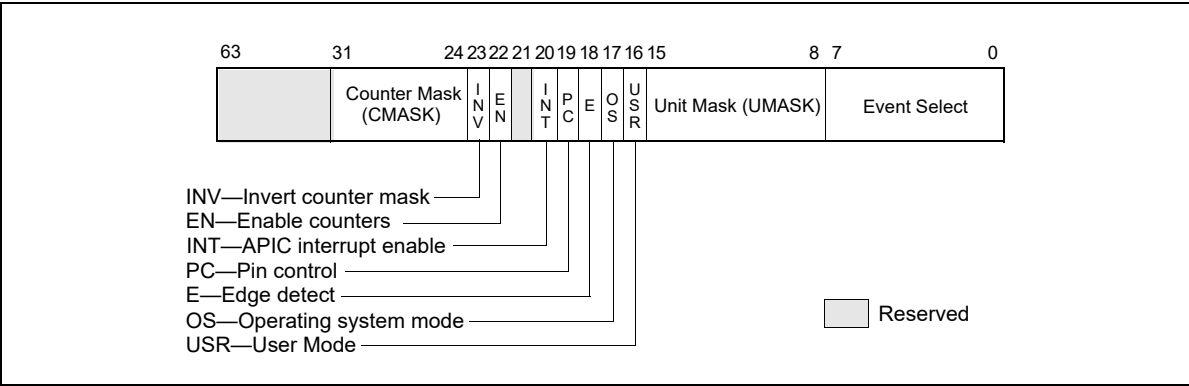


Figure 22-1. Layout of IA32_PERFEVTSELx MSRs

- **Unit mask (UMASK) field (bits 8 through 15)** — These bits qualify the condition that the selected event logic unit detects. Valid UMASK values for each event logic unit are specific to the unit. For each architectural performance event, its corresponding UMASK value defines a specific microarchitectural condition.

A pre-defined microarchitectural condition associated with an architectural event may not be applicable to a given processor. The processor then reports only a subset of pre-defined architectural events. Pre-defined architectural events are listed in Table 22-3; support for pre-defined architectural events is enumerated using CPUID.0AH:EBX.

- **USR (user mode) flag (bit 16)** — Specifies that the selected microarchitectural condition is counted when the logical processor is operating at privilege levels 1, 2 or 3. This flag can be used with the OS flag.
- **OS (operating system mode) flag (bit 17)** — Specifies that the selected microarchitectural condition is counted when the logical processor is operating at privilege level 0. This flag can be used with the USR flag.
- **E (edge detect) flag (bit 18)** — Enables (when set) edge detection of the selected microarchitectural condition. The logical processor counts the number of deasserted to asserted transitions for any condition that can be expressed by the other fields. The mechanism does not permit back-to-back assertions to be distinguished.

This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).

- **PC (pin control) flag (bit 19)** — Beginning with Sandy Bridge microarchitecture, this bit is reserved (not writeable). On processors based on previous microarchitectures, the logical processor toggles the PMi pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PMi pins when the counter overflows. The toggling of a pin is defined as assertion of the pin for a single bus clock followed by deassertion.

- **INT (APIC interrupt enable) flag (bit 20)** — When set, the logical processor generates an exception through its local APIC on counter overflow.
- **EN (Enable Counters) Flag (bit 22)** — When set, performance counting is enabled in the corresponding performance-monitoring counter; when clear, the corresponding counter is disabled. The event logic unit for a UMASK must be disabled by setting `IA32_PERFEVTSELx[bit 22] = 0`, before writing to `IA32_PMCx`.
- **INV (invert) flag (bit 23)** — When set, inverts the counter-mask (CMASK) comparison, so that both greater than or equal to and less than comparisons can be made (0: greater than or equal; 1: less than). Note if counter-mask is programmed to zero, INV flag is ignored.
- **Counter mask (CMASK) field (bits 24 through 31)** — When this field is not zero, a logical processor compares this mask to the events count of the detected microarchitectural condition during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented.

This mask is intended for software to characterize microarchitectural conditions that can count multiple occurrences per cycle (for example, two or more instructions retired per clock; or bus queue occupations). If the counter-mask field is 0, then the counter is incremented each cycle by the event count associated with multiple occurrences.

NOTE

A non-zero write of a field that is introduced in a later architectural performance monitoring version results in a general-protection (#GP) exception if that field is not supported by prior versions.

22.2.2 Architectural Performance Monitoring Version 2

The enhanced features provided by architectural performance monitoring version 2 include the following:

- **Fixed-function performance counter register and associated control register** — Three of the architectural performance events are counted using three fixed-function MSRs (`IA32_FIXED_CTR0` through `IA32_FIXED_CTR2`). Each of the fixed-function PMC can count only one architectural performance event.
Configuring the fixed-function PMCs is done by writing to bit fields in the MSR (`IA32_FIXED_CTR_CTRL`) located at address 38DH. Unlike configuring performance events for general-purpose PMCs (`IA32_PMCx`) via UMASK field in (`IA32_PERFEVTSELx`), configuring, programming `IA32_FIXED_CTR_CTRL` for fixed-function PMCs do not require any UMASK.
- **Simplified event programming** — Most frequent operation in programming performance events are enabling/disabling event counting and checking the status of counter overflows. Architectural performance event version 2 provides three architectural MSRs:
 - `IA32_PERF_GLOBAL_CTRL` allows software to enable/disable event counting of all or any combination of fixed-function PMCs (`IA32_FIXED_CTRx`) or any general-purpose PMCs via a single WRMSR.
 - `IA32_PERF_GLOBAL_STATUS` allows software to query counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single RDMSR.
 - `IA32_PERF_GLOBAL_OVF_CTRL` allows software to clear counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single WRMSR.
- **PMI Overhead Mitigation** — Architectural performance monitoring version 2 introduces two bit field interface in `IA32_DEBUGCTL` for PMI service routine to accumulate performance monitoring data and LBR records with reduced perturbation from servicing the PMI. The two bit fields are:
 - `IA32_DEBUGCTL.Freeze_LBR_On_PMI(bit 11)`. In architectural performance monitoring version 2, only the legacy semantic behavior is supported. See Section 20.4.7 for details of the legacy Freeze LBRs on PMI control.
 - `IA32_DEBUGCTL.Freeze_PerfMon_On_PMI(bit 12)`. In architectural performance monitoring version 2, only the legacy semantic behavior is supported. See Section 20.4.7 for details of the legacy Freeze LBRs on PMI control.

The facilities provided by architectural performance monitoring version 2 can be queried from `CPUID.0AH` by examining the content of register `EDX`:

- Bits 0 through 4 of CPUID.0AH:EDX indicates the number of fixed-function performance counters available per core,
- Bits 5 through 12 of CPUID.0AH:EDX indicates the bit-width of fixed-function performance counters. Bits beyond the width of the fixed-function counter are reserved and must be written as zeros.

NOTE

Early generation of processors based on Intel Core microarchitecture may report in CPUID.0AH:EDX of support for version 2 but indicating incorrect information of version 2 facilities.

The IA32_FIXED_CTR_CTRL MSR include multiple sets of 4-bit field, each 4 bit field controls the operation of a fixed-function performance counter. Figure 22-2 shows the layout of 4-bit controls for each fixed-function PMC. Two sub-fields are currently defined within each control. The definitions of the bit fields are:

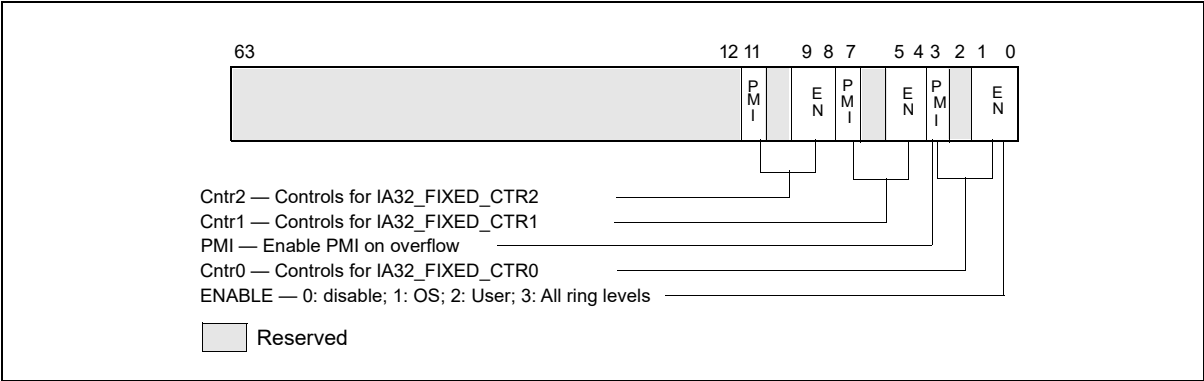


Figure 22-2. Layout of IA32_FIXED_CTR_CTRL MSR

- **Enable field (lowest 2 bits within each 4-bit control)** — When bit 0 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring 0. When bit 1 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring greater than 0. Writing 0 to both bits stops the performance counter. Writing a value of 11B enables the counter to increment irrespective of privilege levels.
- **PMI field (the fourth bit within each 4-bit control)** — When set, the logical processor generates an exception through its local APIC on overflow condition of the respective fixed-function counter.

IA32_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting of each performance counter. Figure 22-3 shows the layout of IA32_PERF_GLOBAL_CTRL. Each enable bit in IA32_PERF_GLOBAL_CTRL is ANDed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_PERF_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the ANDed results is true; counting is disabled when the result is false.

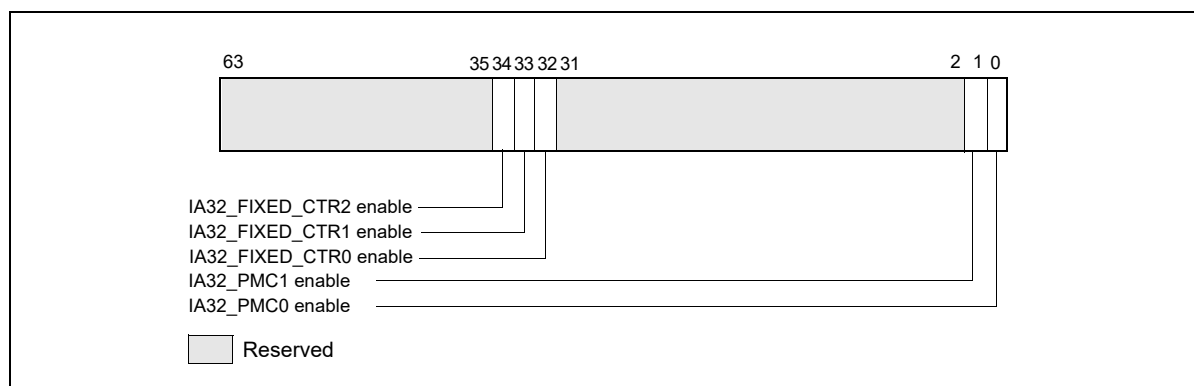


Figure 22-3. Layout of IA32_PERF_GLOBAL_CTRL MSR

The behavior of the fixed function performance counters supported by architectural performance version 2 is expected to be consistent on all processors that support those counters, and is defined as follows.

Table 22-1. Association of Fixed-Function Performance Counters with Architectural Performance Events

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
IA32_FIXED_CTR0	309H	INST_RETIRED.ANY	This event counts the number of instructions that retire execution. For instructions that consist of multiple uops, this event counts the retirement of the last uop of the instruction. The counter continues counting during hardware interrupts, traps, and in-side interrupt handlers.
IA32_FIXED_CTR1	30AH	CPU_CLK_UNHALTED.THREAD CPU_CLK_UNHALTED.CORE	The CPU_CLK_UNHALTED.THREAD event counts the number of core cycles while the logical processor is not in a halt state. If there is only one logical processor in a processor core, CPU_CLK_UNHALTED.CORE counts the unhalted cycles of the processor core. The core frequency may change from time to time due to transitions associated with Enhanced Intel SpeedStep Technology or TM2. For this reason this event may have a changing ratio with regards to time.
IA32_FIXED_CTR2	30BH	CPU_CLK_UNHALTED.REF_TSC	This event counts the number of reference cycles at the TSC rate when the core is not in a halt state and not in a TM stop-clock state. The core enters the halt state when it is running the HLT instruction or the MWAIT instruction. This event is not affected by core frequency changes (e.g., P states) but counts at the same frequency as the time stamp counter. This event can approximate elapsed time while the core was not in a halt state and not in a TM stopclock state.
IA32_FIXED_CTR3	30CH	TOPDOWN.SLOTS	This event counts the number of available slots for an unhalted logical processor. The event increments by machine-width of the narrowest pipeline as employed by the Top-down Microarchitecture Analysis method. The count is distributed among unhalted logical processors (hyper-threads) who share the same physical core. Software can use this event as the denominator for the top-level metrics of the Top-down Microarchitecture Analysis method.

Table 22-1. Association of Fixed-Function Performance Counters with Architectural Performance Events

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
IA32_FIXED_CTR4 ¹	30DH	TOPDOWN_BAD_SPECULATION	This event counts Topdown slots that were not consumed by the backend due to a pipeline flush, such as a mispredicted branch or a machine clear. It provides a value equivalent to a general-purpose counter configured with UMask=00H and EventSelect=73H.
IA32_FIXED_CTR5 ¹	30EH	TOPDOWN_FE_BOUND	This event counts Topdown slots where uops were not provided to the backend due to frontend limitations, such as instruction cache/TLB miss delays or decoder limitations. It provides a value equivalent to a general purpose counter configured with UMask=01H and EventSelect=9CH.
IA32_FIXED_CTR6 ¹	30FH	TOPDOWN_RETIRING	This event counts Topdown slots that were committed (retired) by the backend. It provides a value equivalent to a general purpose counter configured with UMask=02H and EventSelect=C2H.

NOTES:

1. If this counter is supported, it will be accessible in the following MSRs: IA32_PERF_GLOBAL_STATUS (38EH), IA32_PERF_GLOBAL_CTRL (38FH), IA32_PERF_GLOBAL_STATUS_RESET (390H), and IA32_PERF_GLOBAL_STATUS_SET (391H).

IA32_PERF_GLOBAL_STATUS MSR provides single-bit status for software to query the overflow condition of each performance counter. IA32_PERF_GLOBAL_STATUS[bit 62] indicates overflow conditions of the DS area data buffer. IA32_PERF_GLOBAL_STATUS[bit 63] provides a CondChgd bit to indicate changes to the state of performance monitoring hardware. Figure 22-4 shows the layout of IA32_PERF_GLOBAL_STATUS. A value of 1 in bits 0, 1, 32 through 34 indicates a counter overflow condition has occurred in the associated counter.

When a performance counter is configured for PEBS, overflow condition in the counter generates a performance-monitoring interrupt signaling a PEBS event. On a PEBS event, the processor stores data records into the buffer area, clears the counter overflow status., and sets the “OvfBuffer” bit in IA32_PERF_GLOBAL_STATUS.

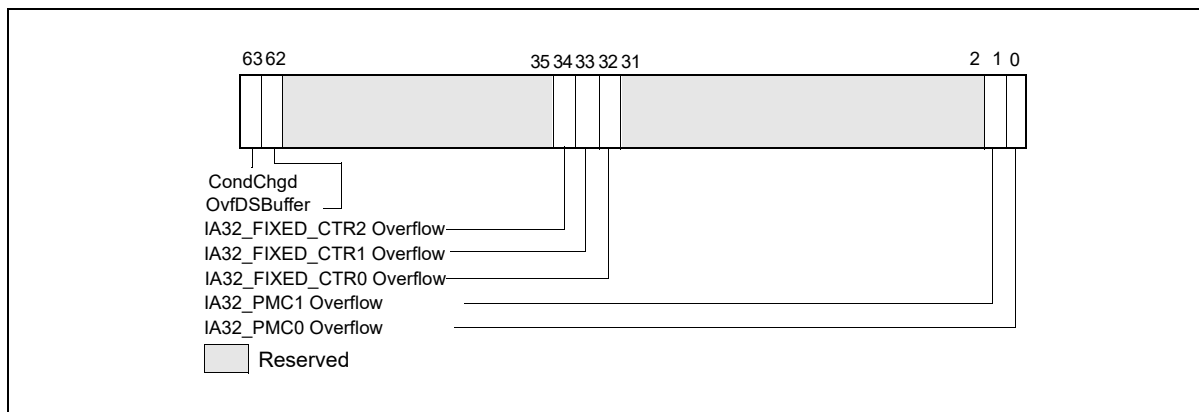


Figure 22-4. Layout of IA32_PERF_GLOBAL_STATUS MSR

IA32_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow indicator(s) of any general-purpose or fixed-function counters via a single WRMSR. Software should clear overflow indications when

- Setting up new values in the event select and/or UMASK field for counting or interrupt-based event sampling.
- Reloading counter values to continue collecting next sample.
- Disabling event counting or interrupt-based event sampling.

The layout of IA32_PERF_GLOBAL_OVF_CTL is shown in Figure 22-5.

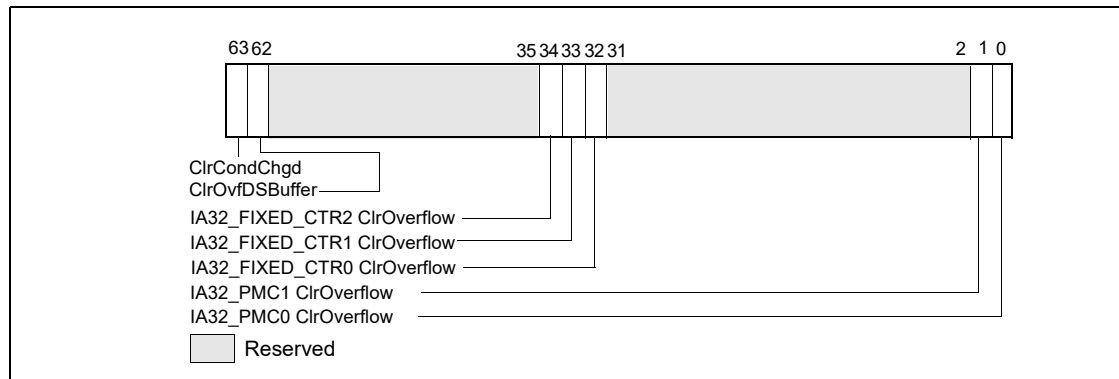


Figure 22-5. Layout of IA32_PERF_GLOBAL_OVF_CTRL MSR

22.2.3 Architectural Performance Monitoring Version 3

Processors supporting architectural performance monitoring version 3 also supports version 1 and 2, as well as capability enumerated by CPUID.0AH. Specifically, version 3 provides the following enhancement in performance monitoring facilities if a processor core comprising of more than one logical processor, i.e., a processor core supporting Intel Hyper-Threading Technology or simultaneous multi-threading capability:

- AnyThread counting for processor core supporting two or more logical processors. The interface that supports AnyThread counting include:
 - Each IA32_PERFEVTSELx MSR (starting at MSR address 186H) support the bit field layout defined in Figure 22-6.

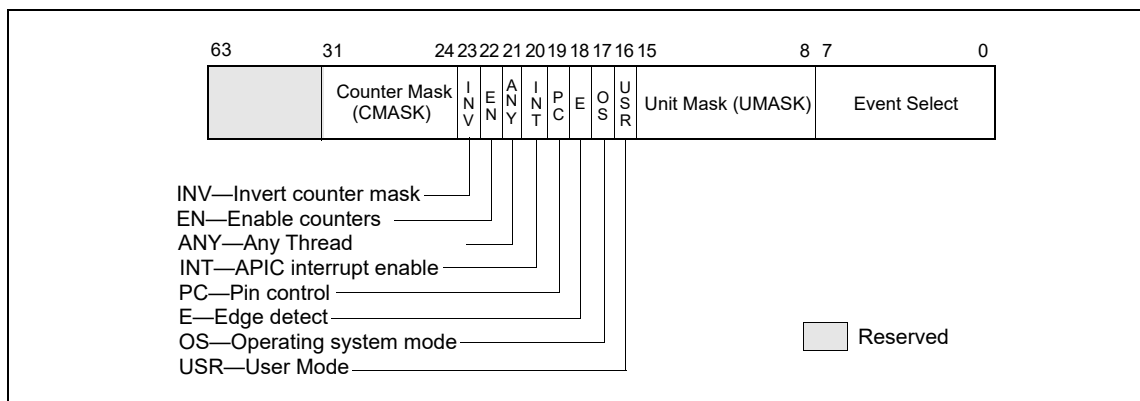


Figure 22-6. Layout of IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

Bit 21 (AnyThread) of IA32_PERFEVTSELx is supported in architectural performance monitoring version 3 for processor core comprising of two or more logical processors. When set to 1, it enables counting the associated event conditions (including matching the thread's CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring across all logical processors sharing a processor core. When bit 21 is 0, the counter only increments the associated event conditions (including matching the thread's CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring in the logical processor which programmed the IA32_PERFEVTSELx MSR.

- Each fixed-function performance counter IA32_FIXED_CTRx (starting at MSR address 309H) is configured by a 4-bit control block in the IA32_PERF_FIXED_CTR_CTRL MSR. The control block also allows thread-specificity configuration using an AnyThread bit for fixed-function counters 0, 1, and 2. The layout of IA32_PERF_FIXED_CTR_CTRL MSR is shown.

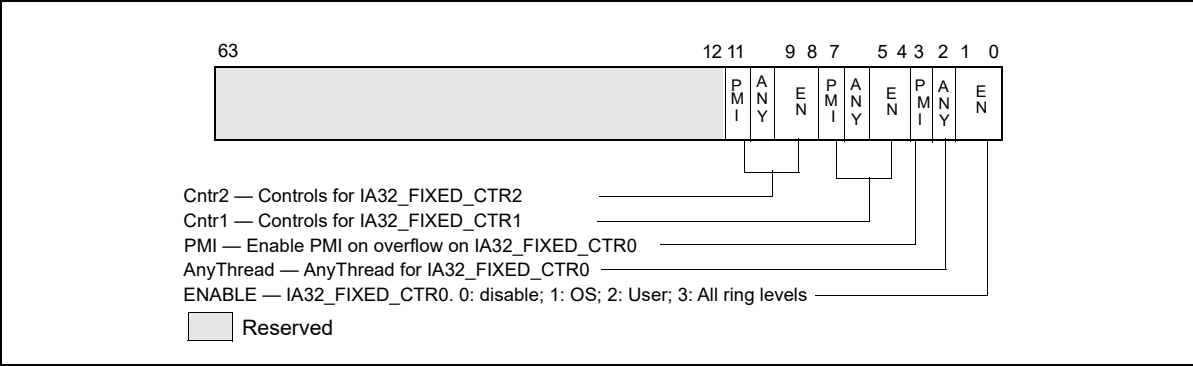


Figure 22-7. IA32_FIXED_CTR_CTRL MSR Supporting Architectural Performance Monitoring Version 3

- Each control block for a fixed-function performance counter provides an **AnyThread** (bit position $2 + 4 \cdot N$, $N = 0, 1$, etc.) bit. When set to 1, it enables counting the associated event conditions (including matching the thread's CPL with the ENABLE setting of the corresponding control block of IA32_PERF_FIXED_CTR_CTRL) occurring across all logical processors sharing a processor core. When an **AnyThread** bit is 0 in IA32_PERF_FIXED_CTR_CTRL, the corresponding fixed-function counter only increments the associated event conditions occurring in the logical processor which programmed the IA32_PERF_FIXED_CTR_CTRL MSR.
- The IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_OVF_CTRL MSRs provide single-bit controls/status for each general-purpose and fixed-function performance counter. Figure 22-8 and Figure 22-9 show the layout of these MSRs for N general-purpose performance counters (where N is reported by CPUID.0AH:EAX[15:8]) and three fixed-function counters.

NOTE

The number of general-purpose performance monitoring counters (i.e., N in Figure 22-9) can vary across processor generations within a processor family, across processor families, or could be different depending on the configuration chosen at boot time in the BIOS regarding Intel Hyper Threading Technology, (e.g., $N=2$ for 45 nm Intel Atom processors; $N=4$ for processors based on the Nehalem microarchitecture; for processors based on the Sandy Bridge microarchitecture, $N=4$ if Intel Hyper Threading Technology is active and $N=8$ if not active). In addition, the number of counters may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters.

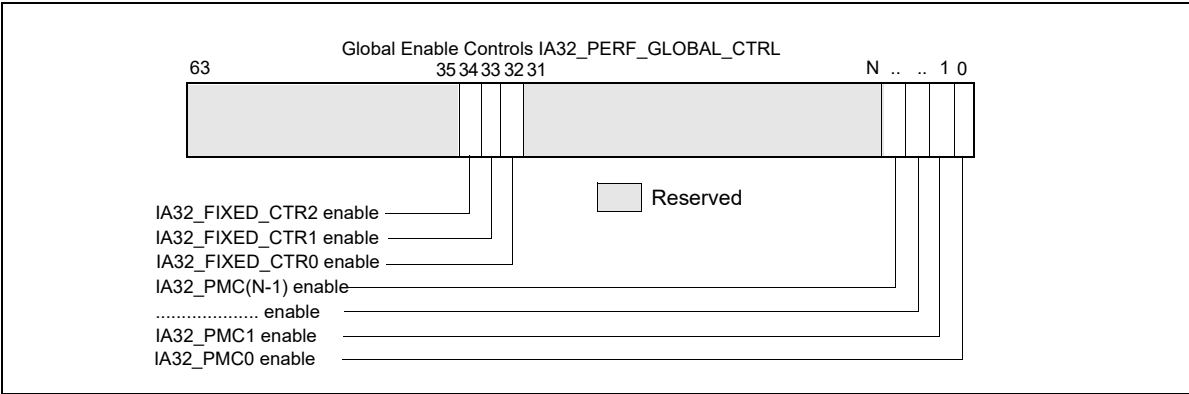


Figure 22-8. Layout of Global Performance Monitoring Control MSR

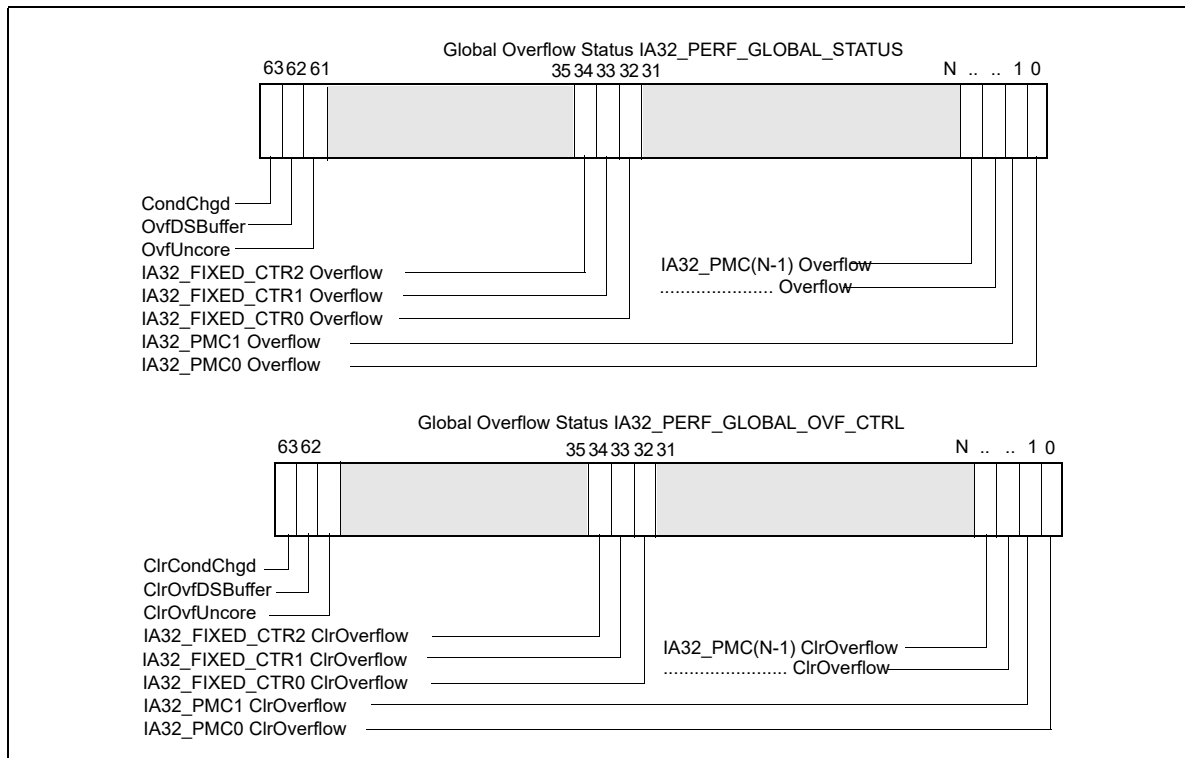


Figure 22-9. Global Performance Monitoring Overflow Status and Control MSRs

22.2.3.1 AnyThread Counting and Software Evolution

The motivation for characterizing software workload over multiple software threads running on multiple logical processors of the same processor core originates from a time earlier than the introduction of the AnyThread interface in IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL. While AnyThread counting provides some benefits in simple software environments of an earlier era, the evolution contemporary software environments introduce certain concepts and pre-requisites that AnyThread counting does not comply with.

One example is the proliferation of software environments that support multiple virtual machines (VM) under VMX (see Chapter 26, “Introduction to Virtual Machine Extensions”) where each VM represents a domain separated from one another.

A Virtual Machine Monitor (VMM) that manages the VMs may allow an individual VM to employ performance monitoring facilities to profiles the performance characteristics of a workload. The use of the Anythread interface in IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL is discouraged with software environments supporting virtualization or requiring domain separation.

Specifically, Intel recommends VMM:

- Configure the MSR bitmap to cause VM-exits for WRMSR to IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL in VMX non-Root operation (see Chapter 27 for additional information),
- Clear the AnyThread bit of IA32_PERFEVTSELx and IA32_FIXED_CTR_CTRL in the MSR-load lists for VM exits and VM entries (see Chapter 27, Chapter 29, and Chapter 30).

Even when operating in simpler legacy software environments which might not emphasize the pre-requisites of a virtualized software environment, the use of the AnyThread interface should be moderated and follow any event-specific guidance where explicitly noted.

22.2.4 Architectural Performance Monitoring Version 4

Processors supporting architectural performance monitoring version 4 also supports version 1, 2, and 3, as well as capability enumerated by CPUID.0AH. Version 4 introduced a streamlined PMI overhead mitigation interface that replaces the legacy semantic behavior but retains the same control interface in IA32_DEBUGCTL.Freeze_LBRs_On_PMI and Freeze_PerfMon_On_PMI. Specifically version 4 provides the following enhancements:

- New indicators (LBR_FRZ, CTR_FRZ) in IA32_PERF_GLOBAL_STATUS, see Section 22.2.4.1.
- Streamlined Freeze/PMI Overhead management interfaces to use IA32_DEBUGCTL.Freeze_LBRs_On_PMI and IA32_DEBUGCTL.Freeze_PerfMon_On_PMI: see Section 22.2.4.1. Legacy semantics of Freeze_LBRs_On_PMI and Freeze_PerfMon_On_PMI (applicable to version 2 and 3) are not supported with version 4 or higher.
- Fine-grain separation of control interface to manage overflow/status of IA32_PERF_GLOBAL_STATUS and read-only performance counter enabling interface in IA32_PERF_GLOBAL_STATUS: see Section 22.2.4.2.
- Performance monitoring resource in-use MSR to facilitate cooperative sharing protocol between perfmon-managing privilege agents.

22.2.4.1 Enhancement in IA32_PERF_GLOBAL_STATUS

The IA32_PERF_GLOBAL_STATUS MSR provides the following indicators with architectural performance monitoring version 4:

- IA32_PERF_GLOBAL_STATUS.LBR_FRZ[bit 58]: This bit is set due to the following conditions:
 - IA32_DEBUGCTL.FREEZE_LBR_ON_PMI has been set by the profiling agent, and
 - A performance counter, configured to generate PMI, has overflowed to signal a PMI. Consequently the LBR stack is frozen.

Effectively, the IA32_PERF_GLOBAL_STATUS.LBR_FRZ bit also serves as a control to enable capturing data in the LBR stack. To enable capturing LBR records, the following expression must hold with architectural PerfMon version 4 or higher:

— $(\text{IA32\_DEBUGCTL.LBR} \& (!\text{IA32\_PERF\_GLOBAL\_STATUS.LBR\_FRZ})) = 1$

- IA32_PERF_GLOBAL_STATUS.CTR_FRZ[bit 59]: This bit is set due to the following conditions:
 - IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI has been set by the profiling agent, and
 - A performance counter, configured to generate PMI, has overflowed to signal a PMI. Consequently, all the performance counters are frozen.

Effectively, the IA32_PERF_GLOBAL_STATUS.CTR_FRZ bit also serve as an read-only control to enable programmable performance counters and fixed counters in the core PMU. To enable counting with the performance counters, the following expression must hold with architectural PerfMon version 4 or higher:

- $(\text{IA32\_PERFEVTSELn.EN} \& \text{IA32\_PERF\_GLOBAL\_CTRL.PMCn} \& (!\text{IA32\_PERF\_GLOBAL\_STATUS.CTR\_FRZ})) = 1$ for programmable counter 'n', or
- $(\text{IA32\_PERF\_FIXED\_CTRL.ENi} \& \text{IA32\_PERF\_GLOBAL\_CTRL.FCi} \& (!\text{IA32\_PERF\_GLOBAL\_STATUS.CTR\_FRZ})) = 1$ for fixed counter 'i'

The read-only enable interface IA32_PERF_GLOBAL_STATUS.CTR_FRZ provides a more efficient flow for a PMI handler to use IA32_DEBUGCTL.Freeze_PerfMon_On_PMI to filter out data that may distort target workload analysis, see Table 20-3. It should be noted the IA32_PERF_GLOBAL_CTRL register continue to serve as the primary interface to control all performance counters of the logical processor.

For example, when the Freeze-On-PMI mode is not being used, a PMI handler would be setting IA32_PERF_GLOBAL_CTRL as the very last step to commence the overall operation after configuring the individual counter registers, controls, and PEBS facility. This does not only assure atomic monitoring but also avoids unnecessary complications (e.g., race conditions) when software attempts to change the core PMU configuration while some counters are kept enabled.

Additionally, IA32_PERF_GLOBAL_STATUS.TraceToPAPMI[bit 55]: On processors that support Intel Processor Trace and configured to store trace output packets to physical memory using the ToPA scheme, bit 55 is set when a PMI occurred due to a ToPA entry memory buffer was completely filled.

IA32_PERF_GLOBAL_STATUS also provides an indicator to distinguish interaction of performance monitoring operations with other side-band activities, which apply Intel SGX on processors that support it (for additional information about Intel SGX, see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D):

- IA32_PERF_GLOBAL_STATUS.ASCI[bit 60]: This bit is set when data accumulated in any of the configured performance counters (i.e., IA32_PMCx or IA32_FIXED_CTRx) may include contributions from direct or indirect operation of Intel SGX to protect an enclave (since the last time IA32_PERF_GLOBAL_STATUS.ASCI was cleared).

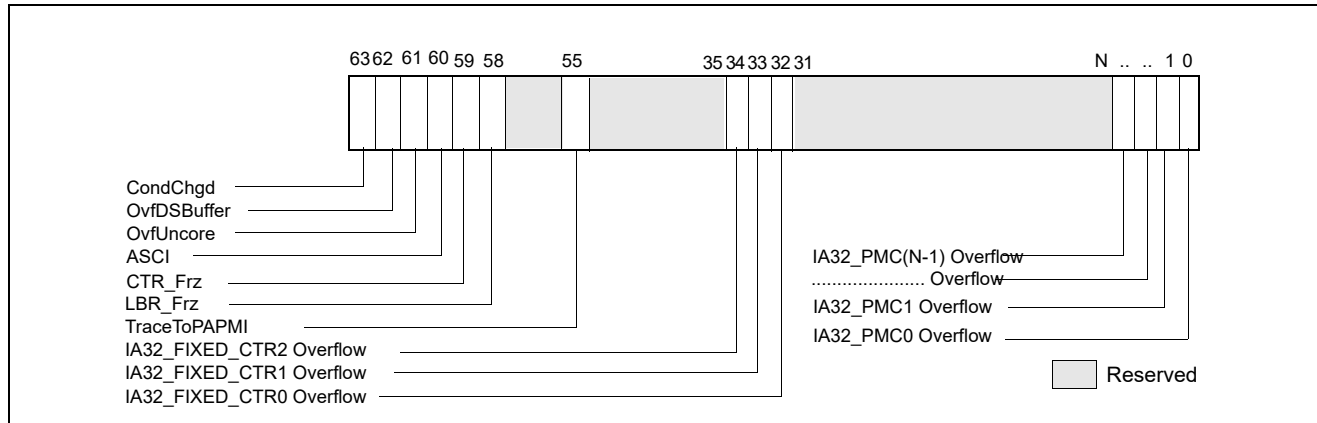


Figure 22-10. IA32_PERF_GLOBAL_STATUS MSR and Architectural PerfMon Version 4

Note, a processor's support for IA32_PERF_GLOBAL_STATUS.TraceToPAPMI[bit 55] is enumerated as a result of CPUID enumerated capability of Intel Processor Trace and the use of the ToPA buffer scheme. Support of IA32_PERF_GLOBAL_STATUS.ASCI[bit 60] is enumerated by the CPUID enumeration of Intel SGX.

22.2.4.2 IA32_PERF_GLOBAL_STATUS_RESET and IA32_PERF_GLOBAL_STATUS_SET MSRS

With architectural performance monitoring version 3 and lower, clearing of the set bits in IA32_PERF_GLOBAL_STATUS MSR by software is done via IA32_PERF_GLOBAL_OVF_CTRL MSR. Starting with architectural performance monitoring version 4, software can manage the overflow and other indicators in IA32_PERF_GLOBAL_STATUS using separate interfaces to set or clear individual bits.

The address and the architecturally-defined bits of IA32_PERF_GLOBAL_OVF_CTRL is inherited by IA32_PERF_GLOBAL_STATUS_RESET (see Figure 22-11). Further, IA32_PERF_GLOBAL_STATUS_RESET provides additional bit fields to clear the new indicators in IA32_PERF_GLOBAL_STATUS described in Section 22.2.4.1.

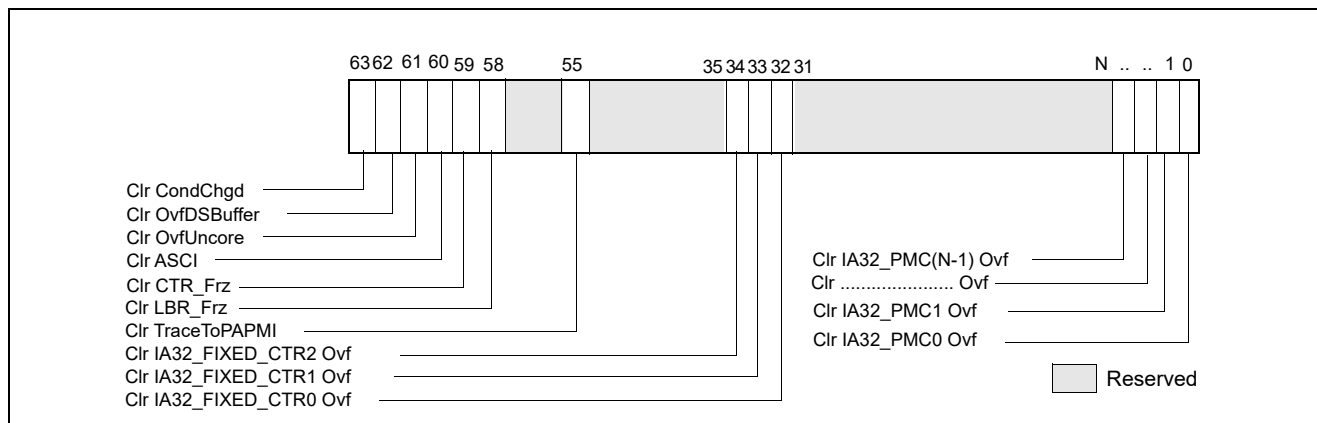


Figure 22-11. IA32_PERF_GLOBAL_STATUS_RESET MSR and Architectural PerfMon Version 4

The IA32_PERF_GLOBAL_STATUS_SET MSR is introduced with architectural performance monitoring version 4. It allows software to set individual bits in IA32_PERF_GLOBAL_STATUS. The IA32_PERF_GLOBAL_STATUS_SET interface can be used by a VMM to virtualize the state of IA32_PERF_GLOBAL_STATUS across VMs.

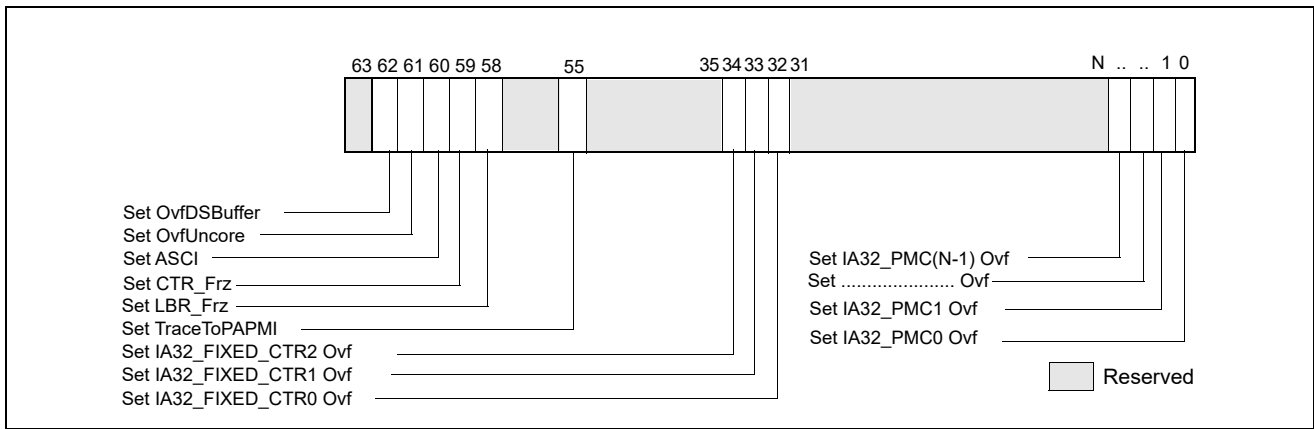


Figure 22-12. IA32_PERF_GLOBAL_STATUS_SET MSR and Architectural PerfMon Version 4

22.2.4.3 IA32_PERF_GLOBAL_INUSE MSR

In a contemporary software environment, multiple privileged service agents may wish to employ the processor's performance monitoring facilities. The IA32_MISC_ENABLE.PERFMON_AVAILABLE[bit 7] interface could not serve the need of multiple agent adequately. A white paper, "Performance Monitoring Unit Sharing Guideline"¹, proposed a cooperative sharing protocol that is voluntary for participating software agents.

Architectural performance monitoring version 4 introduces a new MSR, IA32_PERF_GLOBAL_INUSE, that simplifies the task of multiple cooperating agents to implement the sharing protocol.

The layout of IA32_PERF_GLOBAL_INUSE is shown in Figure 22-13.

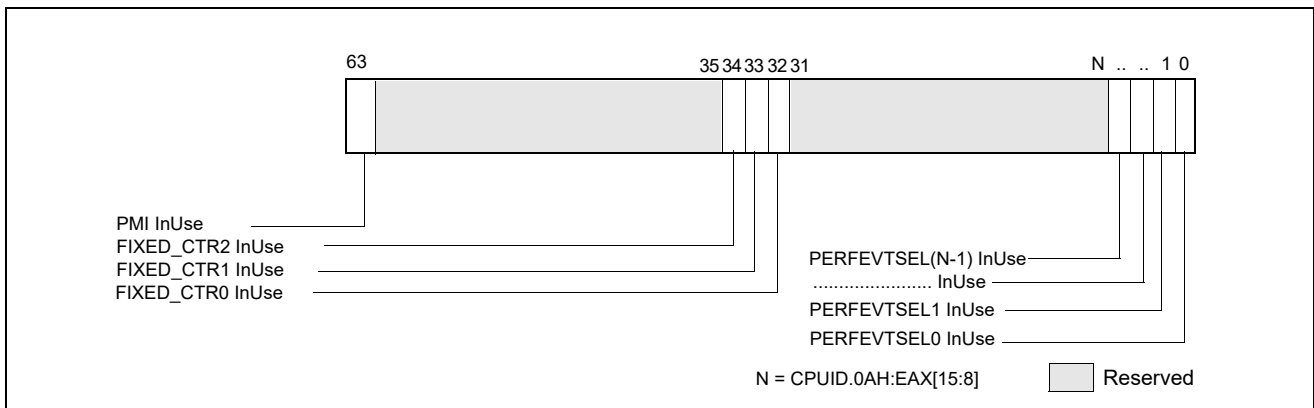


Figure 22-13. IA32_PERF_GLOBAL_INUSE MSR and Architectural PerfMon Version 4

The IA32_PERF_GLOBAL_INUSE MSR provides an "InUse" bit for each programmable performance counter and fixed counter in the processor. Additionally, it includes an indicator if the PMI mechanism has been configured by a profiling agent.

- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL0_InUse[bit 0]: This bit reflects the logical state of (IA32_PERFEVTSEL0[7:0] != 0).

1. Available at <http://www.intel.com/sdm>

- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL1_InUse[bit 1]: This bit reflects the logical state of (IA32_PERFEVTSEL1[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSEL2_InUse[bit 2]: This bit reflects the logical state of (IA32_PERFEVTSEL2[7:0] != 0).
- IA32_PERF_GLOBAL_INUSE.PERFEVTSELn_InUse[bit n]: This bit reflects the logical state of (IA32_PERFEVTSELn[7:0] != 0), n < CPUID.0AH:EAX[15:8].
- IA32_PERF_GLOBAL_INUSE.FC0_InUse[bit 32]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[1:0] != 0).
- IA32_PERF_GLOBAL_INUSE.FC1_InUse[bit 33]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[5:4] != 0).
- IA32_PERF_GLOBAL_INUSE.FC2_InUse[bit 34]: This bit reflects the logical state of (IA32_FIXED_CTR_CTRL[9:8] != 0).
- IA32_PERF_GLOBAL_INUSE.PMI_InUse[bit 63]: This bit is set if any one of the following bit is set:
 - IA32_PERFEVTSELn.INT[bit 20], n < CPUID.0AH:EAX[15:8].
 - IA32_FIXED_CTR_CTRL.ENi_PMI, i = 0, 1, 2.
 - Any IA32_PEBS_ENABLES bit which enables PEBS for a general-purpose or fixed-function performance counter.

22.2.5 Architectural Performance Monitoring Version 5

Processors supporting architectural performance monitoring version 5 also support versions 1, 2, 3, and 4, as well as capability enumerated by CPUID.0AH. Specifically, version 5 provides the following enhancements:

- Deprecation of AnyThread mode, see Section 22.2.5.1.
- Individual enumeration of Fixed counters in CPUID.0AH, see Section 22.2.5.2.
- Domain separation, see Section 22.2.5.3.

22.2.5.1 AnyThread Mode Deprecation

With Architectural Performance Monitoring Version 5, a processor that supports AnyThread mode deprecation is enumerated by CPUID.0AH:EDX[15]. If set, software will not have to follow guidelines in Section 22.2.3.1.

22.2.5.2 Fixed Counter Enumeration

With Architectural Performance Monitoring Version 5, register CPUID.0AH:ECX indicates Fixed Counter enumeration. It is a bit mask which enumerates the supported Fixed Counters in a processor. If bit 'i' is set, it implies that Fixed Counter 'i' is supported. Software is recommended to use the following logic to check if a Fixed Counter is supported on a given processor:

$$\text{FxCtr}[i]_{\text{is\_supported}} := \text{ECX}[i] \mid (\text{EDX}[4:0] > i);$$

22.2.5.3 Domain Separation

When the INV flag in IA32_PERFEVTSELx is used, a counter stops counting when the logical processor exits the C0 ACPI C-state.

22.2.6 Architectural Performance Monitoring Version 6

Processors supporting architectural performance monitoring version 6 also support versions 1, 2, 3, 4, and 5, as well as the capabilities enumerated by CPUID leaves 0AH and 23H. Specifically, version 6 provides the following enhancements:

- PerfMon MSRs Aliasing, see Section 22.2.6.1.

- UnitMask2, see Section 22.2.6.2.
- Equal flag, see Section 22.2.6.3.

22.2.6.1 Performance Monitoring MSR Aliasing

Architectural performance monitoring version 6 includes a new range for the counters' MSRs in the 19xxH address range¹. The new MSR range allows for scaling the number of general-purpose and fixed-function counters beyond the quantities in current products. Additionally, it banks registers of the same counter closer to each other.

All legacy and new counters, i.e., those enumerated in CPUID.23H.01H, will be supported in this new address range. Moving forward, newer counters may be supported in the new address range, but not in the legacy one.

Table 22-2. New Performance Monitoring MSR Naming Details

Register	General Counter n	Fixed Counter m
Counter	IA32_PMC_GPn_CTR	IA32_PMC_FXm_CTR
Event-Select	IA32_PMC_GPn_CFG_A	N/A
Reload Config	IA32_PMC_GPn_CFG_B	IA32_PMC_FXm_CFG_B
Event-Select Extended	IA32_PMC_GPn_CFG_C	IA32_PMC_FXm_CFG_C

An IA32_PMC_GPn_CTR MSR can be used to access the counter value for a GP (general-purpose) counter 'n.' On processors that support CPUID.23H, a general-purpose (GP) counter 'n' that is enumerated in both CPUID.23H and CPUID.0AH can be accessed through either IA32_PMC_GPn_CTR or the legacy MSR addresses (IA32_PMCn, IA32_A_PMCn). In contrast, a counter 'n' that is only enumerated in CPUID.23H can only be accessed through IA32_PMC_GPn_CTR. This guideline also applies to the other MSR aliases described in this section (i.e., IA32_PMC_GPn_CFG_A and IA32_PERFEVTSELn, IA32_PMC_FXm_CTR and IA32_FIXED_CTRm). The IA32_PMC_GPn_CTR MSR address² for counter 'n' is $1900H + 4 * n$, and this MSR has full-width write support.

The IA32_PMC_GPn_CFG_A MSR can be used to access the performance event select register for a GP counter 'n' and is at address³ $1901H + 4 * n$. The reload configuration MSRs for GP counter 'n,' IA32_PMC_GPn_CFG_B, is at MSR address $1902H + 4 * n$. There is no legacy MSR alias to this reload configuration register. Thus, the register only exists when enumerated in CPUID.23H. Similarly, no legacy MSR alias exists for the event-select extended registers, IA32_PMC_GPn_CFG_C, which are at MSR address $1903H + 4 * n$ for GP counter 'n.'

An IA32_PMC_FXm_CTR MSR can be used to access the counter value for a fixed-function counter 'm' if that counter is enumerated in CPUID.23H. The IA32_PMC_FXm_CTR MSR address for fixed-function counter 'm' is $1980H + 4 * m$. There is no alias for the fixed-function counters' reload configuration or event select extended registers (IA32_PMC_FXm_CFG_B at MSR addresses $1982H + 4 * m$ and IA32_PMC_FXm_CFG_C at MSR address $1983H + 4 * m$, respectively).

The available general-purpose and fixed-function counters are reported by CPUID.23H.01H:EAX and CPUID.23H.01H:EBX, respectively.⁴ Note that not all counters enumerated in CPUID.23H may have corresponding IA32_PMC_GPn_CFG_B, IA32_PMC_GPn_CFG_C, IA32_PMC_FXm_CFG_B, or IA32_PMC_FXm_CFG_C MSRs. The enumeration and usage of these MSRs are described in Section 22.11, "Auto Counter Reload." The enumeration in CPUID.23H is true-view, and thus, the enumeration may only be set on (and the MSRs/counters they enumerate only supported on) a subset of the logical processors of the system.

The architectural performance monitoring version 6 enhanced layout of the IA32_PERFEVTSELx MSR is shown in Figure 22-14.

1. This feature is also available in a subset of processors with a CPUID signature value of DisplayFamily_DisplayModel 06_C5H or 06_C6H (though they report IA32_PERF_CAPABILITIES.PEBS_FMT as 5).
2. As an example, the IA32_PMC_GP1_CTR MSR has MSR address 1904H. Note that the legacy full-width MSR addresses for the counters, IA32_A_PMCn MSRs, remains at MSR address $4C1H + n$.
3. As an example, the IA32_PMC_GP1_CFG_A MSR has MSR address 1905H. Note that the legacy MSR address for the event select registers, IA32_PERFEVTSELn MSRs, remain at MSR address $186H + n$.
4. The valid range of fixed-function counters is 0 through 15.

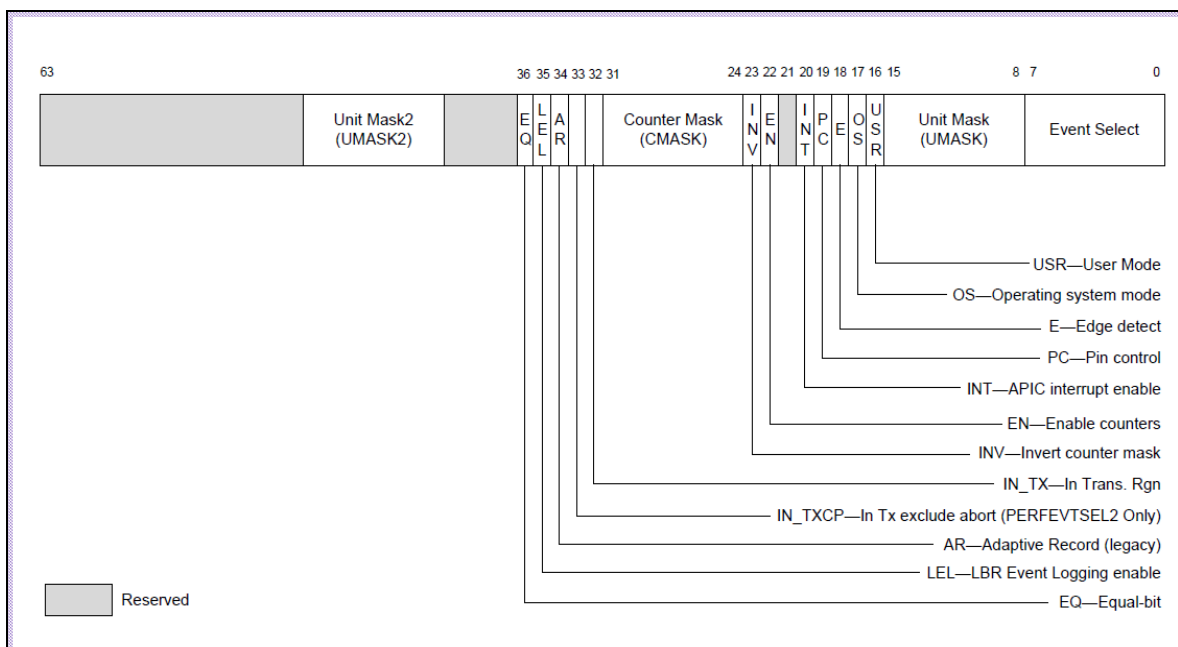


**Figure 22-14. IA32_PMC_GPx_CFG_A MSR (also known as IA32_PERFEVTSELx)
Supporting Architectural Performance Monitoring Version 6**

NOTES

The EQ bit and UMASK2 field are added in Architectural Performance Monitoring Version 6.

The IN_TX and IN_TXCP bits are available only on processors supporting Intel TSX.

The architectural performance monitoring version 6 enhanced layout of the IA32_FIXED_CTR_CTRL MSR is shown in Figure 22-15.

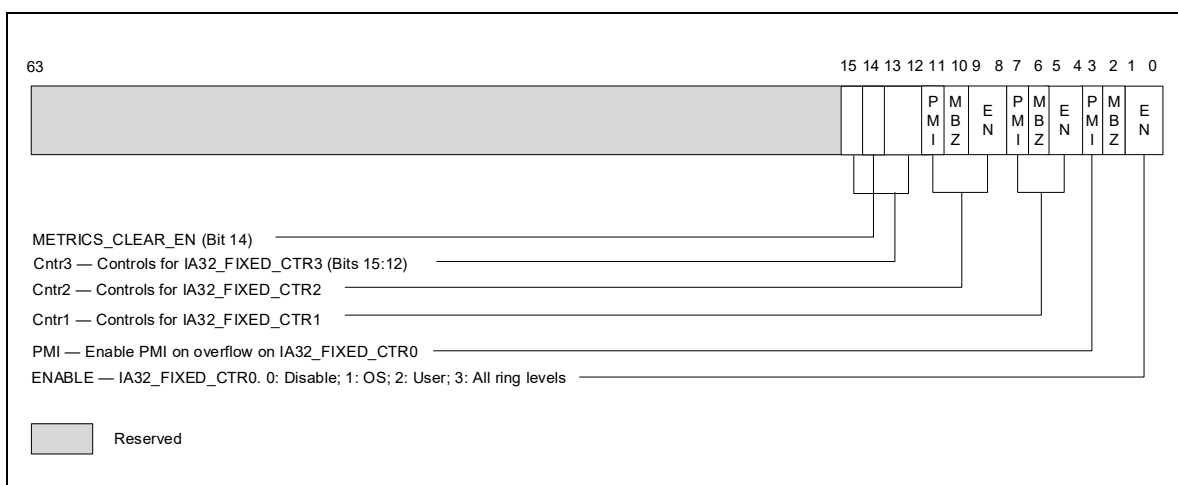


Figure 22-15. IA32_FIXED_CTR_CTRL MSR Supporting Architectural Performance Monitoring Version 6

22.2.6.2 Unit Mask 2

Architectural performance monitoring version 6 introduces a new Unit Mask 2 (UMASK2) field in the IA32_PERFEVTSELx MSRs. It is supported if enumerated by CPUID.23H.00H:EBX[0].

- UMASK2 field (bits 40 through 47): These bits qualify the condition that the selected event logic unit detects. Valid UMASK2 values for each event logic unit are specific to the unit. The new UMASK2 field may also be used in conjunction with UMASK.

22.2.6.3 Equal Flag

Architectural performance monitoring version 6 introduces a new Equal (EQ) flag in the IA32_PERFVTSELx MSRs. It is supported if enumerated by CPUID.23H.00H:EBX[1].

- EQ flag (bit 36): When the EQ flag is set and the INV flag is clear, the comparison evaluates to true if the selected performance monitoring event (the event) is equal to the specified Counter Mask value (CMask). When the EQ flag is set and the INV flag is set, the comparison evaluates to true if the event is less than the CMask value and the event is not zero. Note that if the CMask is zero, the EQ flag is ignored.

22.2.7 Pre-defined Architectural Performance Events

Table Table 22-3. "UMask and Event Select Encodings for Pre-Defined Architectural Performance Events" lists architecturally defined events.

Table 22-3. UMask and Event Select Encodings for Pre-Defined Architectural Performance Events

Bit Position CPUID.0AH:EBX and CPUID.23H.03H:EAX	Event Name	UMask	Event Select
0	UnHalted Core Cycles	00H	3CH
1	Instruction Retired	00H	C0H
2	UnHalted Reference Cycles ¹	01H	3CH
3	LLC Reference	4FH	2EH
4	LLC Misses	41H	2EH
5	Branch Instruction Retired	00H	C4H
6	Branch Misses Retired	00H	C5H
7	Topdown Slots	01H	A4H
8	Topdown Backend Bound	02H	A4H
9	Topdown Bad Speculation	00H	73H
10	Topdown Frontend Bound	01H	9CH
11	Topdown Retiring	02H	C2H
12	LBR Inserts	01H	E4H

NOTES:

1. Implementations prior to the 12th generation Intel® Core™ processor P-cores count at core crystal clock, TSC, or bus clock frequency.

A processor that supports architectural performance monitoring may not support all the predefined architectural performance events (Table Table 22-3. "UMask and Event Select Encodings for Pre-Defined Architectural Performance Events"). The number of architectural events is reported through CPUID.0AH:EAX[31:24], while non-zero bits in CPUID.0AH:EBX indicate any architectural events that are not available.

The behavior of each architectural performance event is expected to be consistent on all processors that support that event. Minor variations between microarchitectures are noted below:

- UnHalted Core Cycles** — Event select 3CH, Umask 00H

This event counts core clock cycles when the clock signal on a specific core is running (not halted). The counter does not advance in the following conditions:

- An ACPI C-state other than C0 for normal operation.

- HLT.
- STPCLK# pin asserted.
- Being throttled by TM1.
- During the frequency switching phase of a performance state transition (see Chapter 17, “Power and Thermal Management”).

The performance counter for this event counts across performance state transitions using different core clock frequencies.

- **Instructions Retired** — Event select C0H, Umask 00H

This event counts the number of instructions at retirement. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. An instruction with a REP prefix counts as one instruction (not per iteration). Faults before the retirement of the last micro-op of a multi-ops instruction are not counted.

This event does not increment under VM-exit conditions. Counters continue counting during hardware interrupts, traps, and inside interrupt handlers.

- **UnHalted Reference Cycles** — Event select 3CH, Umask 01H

This event counts reference clock cycles at a fixed frequency while the clock signal on the core is running. The event counts at a fixed frequency, irrespective of core frequency changes due to performance state transitions. Processors may implement this behavior differently. Current implementations use the core crystal clock, TSC or the bus clock. Because the rate may differ between implementations, software should calibrate it to a time source with known frequency.

- **Last Level Cache References** — Event select 2EH, Umask 4FH

This event counts requests originating from the core that reference a cache line in the last level on-die cache. The event count includes speculation and cache line fills due to the first-level cache hardware prefetcher, but may exclude cache line fills due to other hardware-prefetchers.

Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.

- **Last Level Cache Misses** — Event select 2EH, Umask 41H

This event counts each cache miss condition for references to the last level on-die cache. The event count may include speculation and cache line fills due to the first-level cache hardware prefetcher, but may exclude cache line fills due to other hardware-prefetchers.

Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.

- **Branch Instructions Retired** — Event select C4H, Umask 00H

This event counts branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction.

- **All Branch Mispredict Retired** — Event select C5H, Umask 00H

This event counts mispredicted branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction in the architectural path of execution and experienced misprediction in the branch prediction hardware.

Branch prediction hardware is implementation-specific across microarchitectures; value comparison to estimate performance differences is not recommended.

- **Topdown Slots** — Event select A4H, Umask 01H

This event counts the total number of available slots for an unhalted logical processor.

The event increments by machine-width of the narrowest pipeline as employed by the Top-down Microarchitecture Analysis method. The count is distributed among unhalted logical processors (hyper-threads) who share the same physical core, in processors that support Intel Hyper-Threading Technology.

Software can use this event as the denominator for the top-level metrics of the Top-down Microarchitecture Analysis method.

NOTE

Programming decisions or software precisions on functionality should not be based on the event values or dependent on the existence of performance monitoring events.

- **Topdown Backend Bound** — Event select A4H, Umask 02H

This event counts a subset of the Topdown Slots event that was not consumed by the backend pipeline due to lack of backend resources, as a result of memory subsystem delays, execution unit limitations, or other conditions.

The count may be distributed among unhalted logical processors that share the same physical core, in processors that support Intel® Hyper-Threading Technology.

Software can use this event as the numerator for the Backend Bound metric (or top-level category) of the Topdown Microarchitecture Analysis method.

Software can also derive the Backend Bound Slots using the formula: Backend Bound Slots = (Total Slots - Bad Speculation Slots - Frontend Bound Slots - Retiring Slots).

- **Topdown Bad Speculation** — Event select 73H, Umask 00H

This event counts a subset of the Topdown Slots event that was wasted due to incorrect speculation as a result of incorrect control-flow or data speculation. Common examples include branch mispredictions and memory ordering clears.

The count may be distributed among impacted logical processors that share the same physical core, for some processors that support Intel Hyper-Threading Technology.

Software can use this event as the numerator for the Bad Speculation metric (or top-level category) of the Topdown Microarchitecture Analysis method.

Software can also derive the Bad Speculation Slots using the formula: Bad Speculation Slots = (Total Slots - Backend Bound Slots - Frontend Bound Slots - Retiring Slots).

- **Topdown Frontend Bound** — Event select 9CH, Umask 01H

This event counts a subset of the Topdown Slots event that had no operation delivered to the backend pipeline due to instruction fetch limitations when the backend could have accepted more operations. Common examples include instruction cache misses and x86 instruction decode limitations.

The count may be distributed among unhalted logical processors that share the same physical core, in processors that support Intel Hyper-Threading Technology.

Software can use this event as the numerator for the Frontend Bound metric (or top-level category) of the Topdown Microarchitecture Analysis method.

- **Topdown Retiring** — Event select C2H, Umask 02H

This event counts a subset of the Topdown Slots event that is utilized by operations that eventually get retired (committed) by the processor pipeline. Usually, this event positively correlates with higher performance as measured by the instructions-per-cycle metric.

Software can use this event as the numerator for the Retiring metric (or top-level category) of the Topdown Microarchitecture Analysis method.

- **LBR Inserts** — Event select E4H, Umask 01H

This event counts when an LBR (Last Branch Record) entry is inserted or removed. Inserted means an actual LBR buffer update has occurred, considering LBR configuration and filtering. An LBR entry is removed when a RET instruction is retired in LBR Call-stack mode.

Software may use this event in usages like profile-guided optimization (PGO) for profiling collections across Intel processors and in virtualized environments.

22.2.8 Full-Width Writes to Performance Counter Registers

The general-purpose performance counter registers IA32_PMCx are writable via WRMSR instruction. However, the value written into IA32_PMCx by WRMSR is the signed extended 64-bit value of the EAX[31:0] input of WRMSR.

A processor that supports full-width writes to the general-purpose performance counters enumerated by CPUID.0AH:EAX[15:8] will set IA32_PERF_CAPABILITIES[13] to enumerate its full-width-write capability. See Figure 22-67.

If IA32_PERF_CAPABILITIES.FW_WRITE[bit 13] = 1, each IA32_PMCi is accompanied by a corresponding alias address starting at 4C1H for IA32_A_PMC0.

The bit width of the performance monitoring counters is specified in CPUID.0AH:EAX[23:16].

If IA32_A_PMCi is present, the 64-bit input value (EDX:EAX) of WRMSR to IA32_A_PMCi will cause IA32_PMCi to be updated by:

```
COUNTERWIDTH = CPUID.0AH:EAX[23:16] bit width of the performance monitoring counter
IA32_PMCi[COUNTERWIDTH-1:32] := EDX[COUNTERWIDTH-33:0];
IA32_PMCi[31:0] := EAX[31:0];
EDX[63:COUNTERWIDTH] are reserved
```

22.2.9 Scalable Enumeration Architecture

The CPUID.23H Architectural Performance Monitoring Extended (ARCH_PERFMON_EXT) leaf provides enhanced enumeration of PMU architectural features. Additionally, the IA32_PERF_CAPABILITIES MSR enhances enumeration for PMU non-architectural features.

NOTE

CPUID.0AH continues to report useful attributes, such as architectural performance monitoring version ID and counter width (# bits).

CPUID.23H enhances previous enumeration of PMU capabilities:

- Supports CPUID sub-leaves to accommodate future PMU extensions.
- Exposes true-view resources per logical processor.
- Introduces a bitmap (true-view) enumeration of general-purpose counters availability.
- A bitmap (true-view) enumeration of fixed-function counters availability.
- A bitmap (true-view) enumeration of architectural performance monitoring events.

Processors that support this enhancement set CPUID.07H.01H:EAX.ARCH_PERFMON_EXT[8].

22.2.9.1 CPUID Sub-leaves

CPUID.23H contains additional architectural PMU capabilities. This leaf supports sub-leaves, providing each distinct PMU feature with an individual sub-leaf for enumerating its details.

The availability of sub-leaves is enumerated via CPUID.23H.00H:EAX. For each bit *n* set in this field, sub-leaf *n* under CPUID.23H is supported.

22.2.9.2 Reporting Per Logical Processor

CPUID.23H provides a true-view of per logical processor PMU capabilities. This leaf reports the actual support of the individual logical processor that the CPUID instruction was executed on; this applies to all sub-leaves.

Software must not make assumptions that CPUID.23H would report any value the same on another logical processor. It is required to read CPUID.23H on every logical processor and program that logical processor only according to the values returned by the CPUID.23H directly executed upon it. It is a requirement of software to compare and determine common features between logical processors if required by iterating over each logical processor's CPUID.23H.

Conversely, CPUID.0AH provides a maximum common set of capabilities across logical processors when a feature is not supported by all logical processors.

NOTE

Locating a PMU feature under CPUID.23H alerts software that the feature may not be supported uniformly across all logical processors.

22.2.9.3 General-Purpose Counters Bitmap

CPUID.23H.01H:EAX reports a bitmap for available general-purpose counters. (CPUID.0AH reports only the total number of general-purpose counters.)

This capability enables a virtual-machine monitor to reserve lower-index counters for its own use, while exposing higher-index counters to guest software. This is especially important should the general-purpose counters not be fully homogeneous.

Software should utilize the new bitmap reporting, including for detecting the number of available general-purpose counters. To facilitate this transition, the number of general-purpose counters in CPUID.0AH will not go beyond eight, even if the processor has support for more than eight general-purpose counters.

Note that general-purpose counters that are exclusively enumerated in CPUID.23H.01H:EAX may not support the legacy MSR address range; see Section 22.2.6.1, “Performance Monitoring MSR Aliasing,” for details.

22.2.9.4 Fixed-Function Counters True-View Bitmap

CPUID.23H.01H:EBX reports a bitmap for available fixed-function counters. (CPUID.0AH reports the common number of contiguous fixed-function counters in addition to a common bitmap of fixed-function counters availability.)¹

This capability enables privileged software to expose per logical processor enumeration of fixed-function counters. This is especially important should the fixed-function counters not be available on all logical processors.

Note that programmable counters that are exclusively enumerated in CPUID.23H.01H:EAX may not support the legacy MSR address range; see Section 22.2.6.1, “Performance Monitoring MSR Aliasing,” for details.

22.2.9.5 Architectural Performance Monitoring Events Bitmap

CPUID.23H.03H:EAX provides a true-view of per logical processor available architectural performance monitoring events. For each bit n set in this field, the processor supports Architectural Performance Monitoring Event of index n (positive polarity).

Conversely, CPUID.0AH:EBX provides a maximum common set of architectural performance monitoring events supported by all logical processors, where if bit n is set, it denotes the processor does not necessarily support Architectural Performance Monitoring Event of index n on all logical processors (negative polarity).

22.2.9.6 TMA Slots Per Cycle

CPUID.23H.00H:ECX[7:0] reports the number of TMA slots per cycle in a true-view per logical-processor fashion.

This number can be multiplied by the number of cycles (from CPU_CLK_UNHALTED.THREAD / CPU_CLK_UNHALTED.CORE or IA32_FIXED_CTR1) to determine the total number of TMA slots.

Because of microarchitectural reasons, some logical processors may be reporting TMA slots per cycle as 0. In such situations, software can use other methods, like programmable events or fixed counters, to understand the performance issues.

1. The valid range of fixed-function counters is 0 through 15.

22.3 PERFORMANCE MONITORING (INTEL® CORE™ PROCESSORS AND INTEL® XEON® PROCESSORS)

22.3.1 Performance Monitoring for Processors Based on Nehalem Microarchitecture

Intel Core i7 processor family¹ supports architectural performance monitoring capability with version ID 3 (see Section 22.2.3) and a host of non-architectural monitoring capabilities. The Intel Core i7 processor family is based on Nehalem microarchitecture, and provides four general-purpose performance counters (IA32_PMC0, IA32_PMC1, IA32_PMC2, IA32_PMC3) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2) in the processor core.

Non-architectural performance monitoring in Intel Core i7 processor family uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>. Non-architectural performance monitoring events fall into two broad categories:

- Performance monitoring events in the processor core: These include many events that are similar to performance monitoring events available to processor based on Intel Core microarchitecture. Additionally, there are several enhancements in the performance monitoring capability for detecting microarchitectural conditions in the processor core or in the interaction of the processor core to the off-core sub-systems in the physical processor package. The off-core sub-systems in the physical processor package is loosely referred to as “uncore”.
- Performance monitoring events in the uncore: The uncore sub-system is shared by more than one processor cores in the physical processor package. It provides additional performance monitoring facility outside of IA32_PMCx and performance monitoring events that are specific to the uncore sub-system.

Architectural and non-architectural performance monitoring events in Intel Core i7 processor family support thread qualification using bit 21 of IA32_PERFEVTSELx MSR.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 22-6 and described in Section and Section 22.2.3.

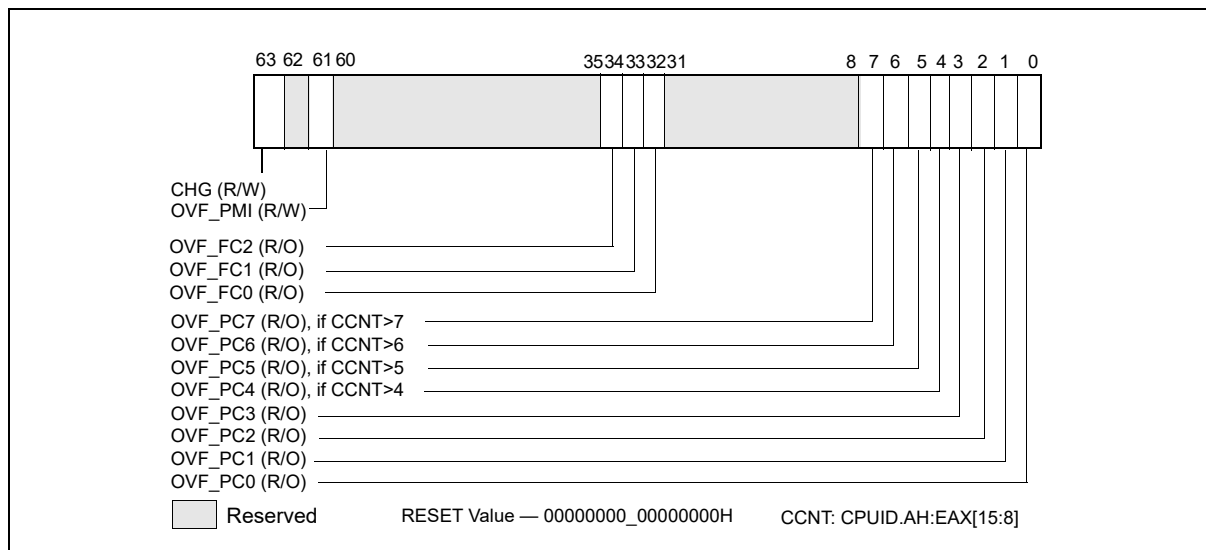


Figure 22-16. IA32_PERF_GLOBAL_STATUS MSR

1. Intel Xeon processor 5500 series and 3400 series are also based on Nehalem microarchitecture; the performance monitoring facilities described in this section generally also apply.

22.3.1.1 Enhancements of Performance Monitoring in the Processor Core

The notable enhancements in the monitoring of performance events in the processor core include:

- Four general purpose performance counters, IA32_PMCx, associated counter configuration MSRs, IA32_PERFVTSELx, and global counter control MSR supporting simplified control of four counters. Each of the four performance counter can support processor event based sampling (PEBS) and thread-qualification of architectural and non-architectural performance events. Width of IA32_PMCx supported by hardware has been increased. The width of counter reported by CPUID.0AH:EAX[23:16] is 48 bits. The PEBS facility in Nehalem microarchitecture has been enhanced to include new data format to capture additional information, such as load latency.
- Load latency sampling facility. Average latency of memory load operation can be sampled using load-latency facility in processors based on Nehalem microarchitecture. This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches). This facility is used in conjunction with the PEBS facility.
- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor core to sub-systems outside the processor core (uncore). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFVTSELx.

NOTE

The number of counters available to software may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may not expose all counters. CPUID.0AH:EAX[15:8] reports the MSRs available to software; see Section 22.2.1.

22.3.1.1.1 Processor Event Based Sampling (PEBS)

All general-purpose performance counters, IA32_PMCx, can be used for PEBS if the performance event supports PEBS. Software uses IA32_MISC_ENABLE[7] and IA32_MISC_ENABLE[12] to detect whether the performance monitoring facility and PEBS functionality are supported in the processor. The MSR IA32_PEBS_ENABLE provides 4 bits that software must use to enable which IA32_PMCx overflow condition will cause the PEBS record to be captured.

Additionally, the PEBS record is expanded to allow latency information to be captured. The MSR IA32_PEBS_ENABLE provides 4 additional bits that software must use to enable latency data recording in the PEBS record upon the respective IA32_PMCx overflow condition. The layout of IA32_PEBS_ENABLE for processors based on Nehalem microarchitecture is shown in Figure 22-17.

When a counter is enabled to capture machine state (PEBS_EN_PMCx = 1), the processor will write machine state information to a memory buffer specified by software as detailed below. When the counter IA32_PMCx overflows from maximum count to zero, the PEBS hardware is armed.

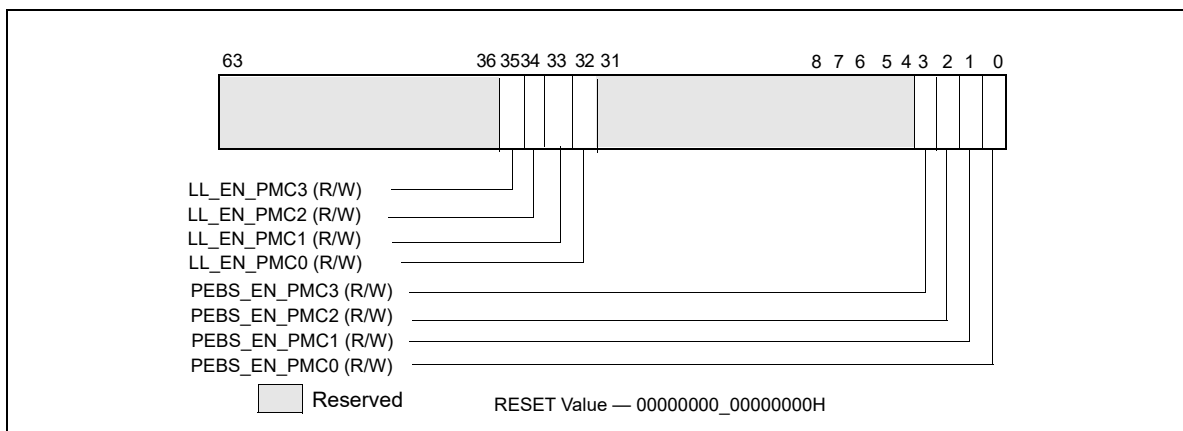


Figure 22-17. Layout of IA32_PEBs_ENABLE MSR

Upon occurrence of the next PEBS event, the PEBS hardware triggers an assist and causes a PEBS record to be written. The format of the PEBS record is indicated by the bit field IA32_PERF_CAPABILITIES[11:8] (see Figure 22-67).

The behavior of PEBS assists is reported by IA32_PERF_CAPABILITIES[6] (see Figure 22-67). The return instruction pointer (RIP) reported in the PEBS record will point to the instruction after (+1) the instruction that causes the PEBS assist. The machine state reported in the PEBS record is the machine state after the instruction that causes the PEBS assist is retired. For instance, if the instructions:

```
mov eax, [eax] ; causes PEBS assist
```

```
nop
```

are executed, the PEBS record will report the address of the nop, and the value of EAX in the PEBS record will show the value read from memory, not the target address of the read operation.

The PEBS record format is shown in Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family", and each field in the PEBS record is 64 bits long. The PEBS record format, along with debug/store area storage format, does not change regardless of IA-32e mode is active or not. CPUID.01H:ECX.DTES64[2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 22-4. PEBS Record Format for Intel Core i7 Processor Family

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	58H	R9
08H	R/EIP	60H	R10
10H	R/EAX	68H	R11
18H	R/EBX	70H	R12
20H	R/ECX	78H	R13
28H	R/EDX	80H	R14
30H	R/ESI	88H	R15
38H	R/EDI	90H	IA32_PERF_GLOBAL_STATUS
40H	R/EBP	98H	Data Linear Address
48H	R/ESP	A0H	Data Source Encoding
50H	R8	A8H	Latency value (core cycles)

In IA-32e mode, the full 64-bit value is written to the register. If the processor is not operating in IA-32e mode, 32-bit value is written to registers with bits 63:32 zeroed. Registers not defined when the processor is not in IA-32e mode are written to zero.

Bytes AFH:90H are enhancement to the PEBS record format. Support for this enhanced PEBS record format is indicated by IA32_PERF_CAPABILITIES[11:8] encoding of 0001B.

The value written to bytes 97H:90H is the state of the IA32_PERF_GLOBAL_STATUS register before the PEBS assist occurred. This value is written so software can determine which counters overflowed when this PEBS record was written. Note that this field indicates the overflow status for all counters, regardless of whether they were programmed for PEBS or not.

Programming PEBS Facility

Only a subset of non-architectural performance events in the processor support PEBS. The subset of precise events are listed in Table 22-88, "PEBS Performance Events for Intel Core Microarchitecture". In addition to using IA32_PERFEVTSELx to specify event unit/mask settings and setting the EN_PMCx bit in the IA32_PEBS_ENABLE register for the respective counter, the software must also initialize the DS_BUFFER_MANAGEMENT_AREA data structure in memory to support capturing PEBS records for precise events.

The recording of PEBS records may not operate properly if accesses to the linear addresses in the DS buffer management area or in the PEBS buffer (see below) cause page faults, VM exits, or the setting of accessed or dirty flags in the paging structures (ordinary or EPT). For that reason, system software should establish paging structures (both ordinary and EPT) to prevent such occurrences. Implications of this may be that an operating system should allocate this memory from a non-paged pool and that system software cannot do "lazy" page-table entry propagation for these pages. A virtual-machine monitor may choose to allow use of PEBS by guest software only if EPT maps all guest-physical memory as present and read/write.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

The beginning linear address of the DS_BUFFER_MANAGEMENT_AREA data structure must be programmed into the IA32_DS_AREA register. The layout of the DS_BUFFER_MANAGEMENT_AREA is shown in Figure 22-18.

- **PEBS Buffer Base:** This field is programmed with the linear address of the first byte of the PEBS buffer allocated by software. The processor reads this field to determine the base address of the PEBS buffer.
- **PEBS Index:** This field is initially programmed with the same value as the PEBS Buffer Base field, or the beginning linear address of the PEBS buffer. The processor reads this field to determine the location of the next PEBS record to write to. After a PEBS record has been written, the processor also updates this field with the address of the next PEBS record to be written. The figure above illustrates the state of PEBS Index after the first PEBS record is written.
- **PEBS Absolute Maximum:** This field represents the absolute address of the maximum length of the allocated PEBS buffer plus the starting address of the PEBS buffer. The processor will not write any PEBS record beyond the end of PEBS buffer, when **PEBS Index** equals **PEBS Absolute Maximum**. No signaling is generated when PEBS buffer is full. Software must reset the **PEBS Index** field to the beginning of the PEBS buffer address to continue capturing PEBS records.

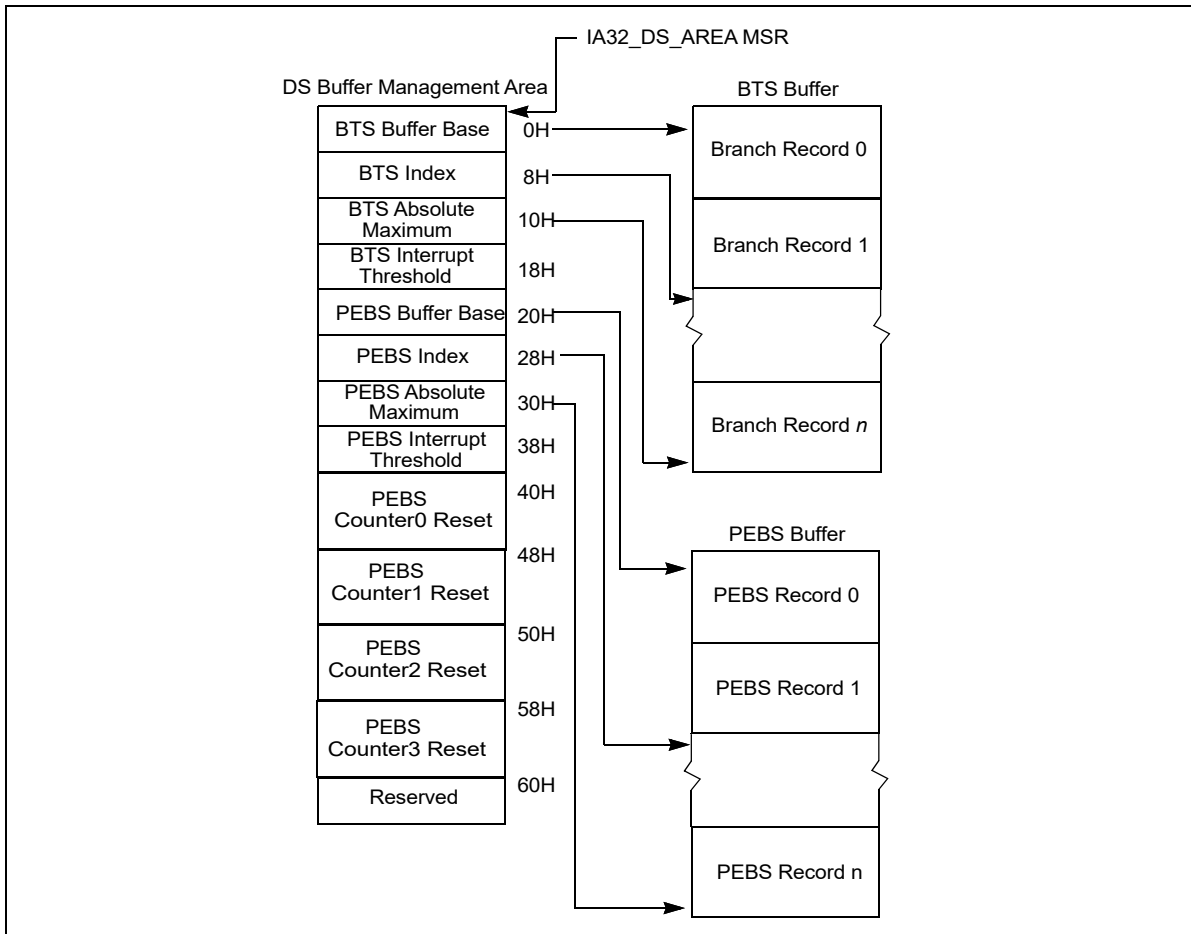


Figure 22-18. PEBS Programming Environment

- PEBS Interrupt Threshold:** This field specifies the threshold value to trigger a performance interrupt and notify software that the PEBS buffer is nearly full. This field is programmed with the linear address of the first byte of the PEBS record within the PEBS buffer that represents the threshold value of this field. After the processor writes a PEBS record and updates **PEBS Index**, if the **PEBS Index** reaches the threshold value of this field, the processor will generate a performance interrupt. This is the same interrupt that is generated by a performance counter overflow, as programmed in the Performance Monitoring Counters vector in the Local Vector Table of the Local APIC. When a performance interrupt due to PEBS buffer full is generated, the `IA32_PERF_GLOBAL_STATUS.PEBS_Ovf` bit will be set.
- PEBS CounterX Reset:** This field allows software to set up PEBS counter overflow condition to occur at a rate useful for profiling workload, thereby generating multiple PEBS records to facilitate characterizing the profile the execution of test code. After each PEBS record is written, the processor checks each counter to see if it overflowed and was enabled for PEBS (the corresponding bit in `IA32_PEBS_ENABLED` was set). If these conditions are met, then the reset value for each overflowed counter is loaded from the DS Buffer Management Area. For example, if counter `IA32_PMC0` caused a PEBS record to be written, then the value of "PEBS Counter 0 Reset" would be written to counter `IA32_PMC0`. If a counter is not enabled for PEBS, its value will not be modified by the PEBS assist.

Performance Counter Prioritization

Performance monitoring interrupts are triggered by a counter transitioning from maximum count to zero (assuming `IA32_PerfEvtSelX.INT` is set). This same transition will cause PEBS hardware to arm, but not trigger. PEBS hardware triggers upon detection of the first PEBS event after the PEBS hardware has been armed (a 0 to 1 transition of the counter). At this point, a PEBS assist will be undertaken by the processor.

Performance counters (fixed and general-purpose) are prioritized in index order. That is, counter IA32_PMC0 takes precedence over all other counters. Counter IA32_PMC1 takes precedence over counters IA32_PMC2 and IA32_PMC3, and so on. This means that if simultaneous overflows or PEBS assists occur, the appropriate action will be taken for the highest priority performance counter. For example, if IA32_PMC1 cause an overflow interrupt and IA32_PMC2 causes an PEBS assist simultaneously, then the overflow interrupt will be serviced first.

The PEBS threshold interrupt is triggered by the PEBS assist, and is by definition prioritized lower than the PEBS assist. Hardware will not generate separate interrupts for each counter that simultaneously overflows. General-purpose performance counters are prioritized over fixed counters.

If a counter is programmed with a precise (PEBS-enabled) event and programmed to generate a counter overflow interrupt, the PEBS assist is serviced before the counter overflow interrupt is serviced. If in addition the PEBS interrupt threshold is met, the

threshold interrupt is generated after the PEBS assist completes, followed by the counter overflow interrupt (two separate interrupts are generated).

Uncore counters may be programmed to interrupt one or more processor cores (see Section 22.3.1.2). It is possible for interrupts posted from the uncore facility to occur coincident with counter overflow interrupts from the processor core. Software must check core and uncore status registers to determine the exact origin of counter overflow interrupts.

22.3.1.1.2 Load Latency Performance Monitoring Facility

The load latency facility provides software a means to characterize the average load latency to different levels of cache/memory hierarchy. This facility requires processor supporting enhanced PEBS record format in the PEBS buffer, see Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family". This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches).

To use this feature software must assure:

- One of the IA32_PERFEVTSELx MSR is programmed to specify the event unit MEM_INST_RETIRED, and the LATENCY_ABOVE_THRESHOLD event mask must be specified (IA32_PerfEvtSelX[15:0] = 100H). The corresponding counter IA32_PMCx will accumulate event counts for architecturally visible loads which exceed the programmed latency threshold specified separately in a MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.
- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with latencies greater than this value are eligible for counting and latency data reporting. The minimum value that may be programmed in this register is 3 (the minimum detectable load latency is 4 core clock cycles).
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register. This means that both the PEBS_EN_CTRX and LL_EN_CTRX bits must be set for the counter(s) of interest. For example, to enable load latency on counter IA32_PMC0, the IA32_PEBS_ENABLE register must be programmed with the 64-bit value 00000001_00000001H.

When the load-latency facility is enabled, load operations are randomly selected by hardware and tagged to carry information related to data source locality and latency. Latency and data source information of tagged loads are updated internally.

When a PEBS assist occurs, the last update of latency and data source information are captured by the assist and written as part of the PEBS record. The PEBS sample after value (SAV), specified in PEBS CounterX Reset, operates orthogonally to the tagging mechanism. Loads are randomly tagged to collect latency data. The SAV controls the number of tagged loads with latency information that will be written into the PEBS record field by the PEBS assists. The load latency data written to the PEBS record will be for the last tagged load operation which retired just before the PEBS assist was invoked.

The load-latency information written into a PEBS record (see Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family", bytes AFH:98H) consists of:

- **Data Linear Address:** This is the linear address of the target of the load operation.

- **Latency Value:** This is the elapsed cycles of the tagged load operation between dispatch to GO, measured in processor core clock domain.
- **Data Source:** The encoded value indicates the origin of the data obtained by the load instruction. The encoding is shown in Table 22-5. "Data Source Encoding for Load Latency Record". In the descriptions, local memory refers to system memory physically attached to a processor package, and remote memory refers to system memory physically attached to another processor package.

Table 22-5. Data Source Encoding for Load Latency Record

Encoding	Description
00H	Unknown L3 cache miss.
01H	Minimal latency core cache hit. This request was satisfied by the L1 data cache.
02H	Pending core cache HIT. Outstanding core cache miss to same cache-line address was already underway.
03H	This data request was satisfied by the L2.
04H	L3 HIT. Local or Remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
05H	L3 HIT. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found. (clean).
06H	L3 HIT. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found.
07H ¹	Reserved/LLC Snoop HitM. Local or Remote home requests that hit the last level cache and were serviced by another core with a cross core snoop where modified copies were found.
08H	Reserved/L3 MISS. Local homed requests that missed the L3 cache and were serviced by forwarded data following a cross package snoop where no modified copies were found. (Remote home requests are not counted).
09H	Reserved
0AH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to shared state).
0BH	L3 MISS. Remote home requests that missed the L3 cache and were serviced by remote DRAM (go to shared state).
0CH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to exclusive state).
0DH	L3 MISS. Remote home requests that missed the L3 cache and were serviced by remote DRAM (go to exclusive state).
0EH	I/O, Request of input/output operation.
0FH	The request was to uncacheable memory.

NOTES:

1. Bit 7 is supported only for processors with a CPUID DisplayFamily_DisplayModel signature of 06_2A, and 06_2E; otherwise it is reserved.

The layout of MSR_PEBS_LD_LAT_THRESHOLD is shown in Figure 22-19.

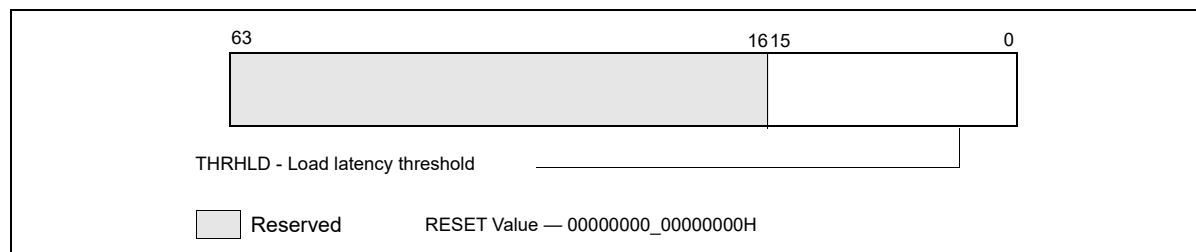


Figure 22-19. Layout of MSR_PEBS_LD_LAT MSR

Bits 15:0 specifies the threshold load latency in core clock cycles. Performance events with latencies greater than this value are counted in IA32_PMCx and their latency information is reported in the PEBS record. Otherwise, they are ignored. The minimum value that may be programmed in this field is 3.

22.3.1.1.3 Off-core Response Performance Monitoring in the Processor Core

Programming a performance event using the off-core response facility can choose any of the four IA32_PERFEVTSELx MSR with specific event codes and predefine mask bit value. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_0. There is only one off-core response configuration MSR. Table Table 22-6. "Off-Core Response Event Encoding" lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 22-6. Off-Core Response Event Encoding

Event code in IA32_PERFEVTSELx	Mask Value in IA32_PERFEVTSELx	Required Off-core Response MSR
B7H	01H	MSR_OFFCORE_RSP_0 (address 1A6H)

The layout of MSR_OFFCORE_RSP_0 is shown in Figure 22-20. Bits 7:0 specifies the request type of a transaction request to the uncore. Bits 15:8 specifies the response of the uncore subsystem.

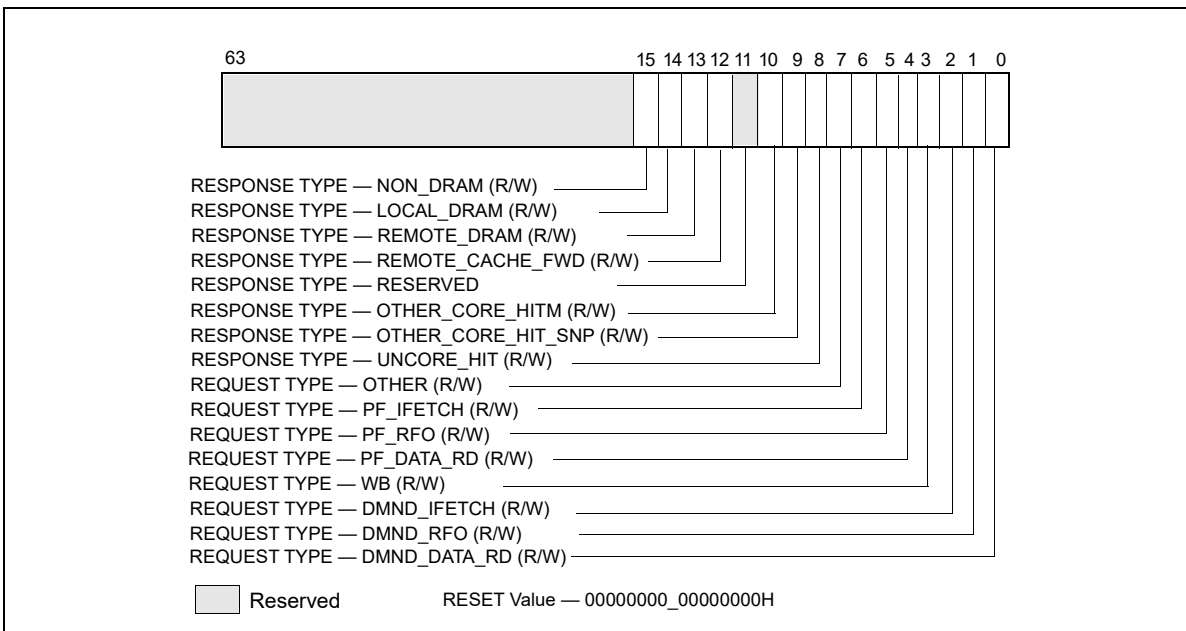


Figure 22-20. Layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 to Configure Off-core Response Events

Table 22-7. MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 Bit Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand and DCU prefetch data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.

Table 22-7. MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 Bit Field Definition (Contd.)

Bit Name	Offset	Description
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
OTHER	7	Counts one of the following transaction types, including L3 invalidate, I/O, full or partial writes, WC or non-temporal stores, CLFLUSH, Fences, lock, unlock, split lock.
UNCORE_HIT	8	L3 Hit: local or remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
OTHER_CORE_HIT_SNP	9	L3 Hit: local or remote home requests that hit L3 cache in the uncore and was serviced by another core with a cross core snoop where no modified copies were found (clean).
OTHER_CORE_HIT_TM	10	L3 Hit: local or remote home requests that hit L3 cache in the uncore and was serviced by another core with a cross core snoop where modified copies were found (HITM).
Reserved	11	Reserved
REMOTE_CACHE_FWD	12	L3 Miss: local homed requests that missed the L3 cache and was serviced by forwarded data following a cross package snoop where no modified copies found. (Remote home requests are not counted)
REMOTE_DRAM	13	L3 Miss: remote home requests that missed the L3 cache and were serviced by remote DRAM.
LOCAL_DRAM	14	L3 Miss: local home requests that missed the L3 cache and were serviced by local DRAM.
NON_DRAM	15	Non-DRAM requests that were serviced by IOH.

22.3.1.2 Performance Monitoring Facility in the Uncore

The “uncore” in Nehalem microarchitecture refers to subsystems in the physical processor package that are shared by multiple processor cores. Some of the sub-systems in the uncore include the L3 cache, Intel QuickPath Interconnect link logic, and integrated memory controller. The performance monitoring facilities inside the uncore operates in the same clock domain as the uncore (U-clock domain), which is usually different from the processor core clock domain. The uncore performance monitoring facilities described in this section apply to Intel Xeon processor 5500 series and processors with the following CUID signatures: 06_1AH, 06_1EH, 06_1FH (see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4). An overview of the uncore performance monitoring facilities is described separately.

The performance monitoring facilities available in the U-clock domain consist of:

- Eight General-purpose counters (MSR_UNCORE_PerfCntr0 through MSR_UNCORE_PerfCntr7). The counters are 48 bits wide. Each counter is associated with a configuration MSR, MSR_UNCORE_PerfEvtSelx, to specify event code, event mask and other event qualification fields. A set of global uncore performance counter enabling/overflow/status control MSRs are also provided for software.
- Performance monitoring in the uncore provides an address/opcode match MSR that provides event qualification control based on address value or QPI command opcode.
- One fixed-function counter, MSR_UNCORE_FixedCntr0. The fixed-function uncore counter increments at the rate of the U-clock when enabled.

The frequency of the uncore clock domain can be determined from the uncore clock ratio which is available in the PCI configuration space register at offset C0H under device number 0 and Function 0.

22.3.1.2.1 Uncore Performance Monitoring Management Facility

MSR_UNCORE_PERF_GLOBAL_CTRL provides bit fields to enable/disable general-purpose and fixed-function counters in the uncore. Figure 22-21 shows the layout of MSR_UNCORE_PERF_GLOBAL_CTRL for an uncore that is shared by four processor cores in a physical package.

- EN_PCn (bit n, n = 0, 7): When set, enables counting for the general-purpose uncore counter MSR_UNCORE_PerfCntr n.

- EN_FC0 (bit 32): When set, enables counting for the fixed-function uncore counter MSR_UNCORE_FixedCnt0.
- EN_PMI_COREn (bit n, n = 0, 3 if four cores are present): When set, processor core n is programmed to receive an interrupt signal from any interrupt enabled uncore counter. PMI delivery due to an uncore counter overflow is enabled by setting IA32_DEBUGCTL.Offcore_PMI_EN to 1.
- PMI_FRZ (bit 63): When set, all U-clock uncore counters are disabled when any one of them signals a performance interrupt. Software must explicitly re-enable the counter by setting the enable bits in MSR_UNCORE_PERF_GLOBAL_CTRL upon exit from the ISR.

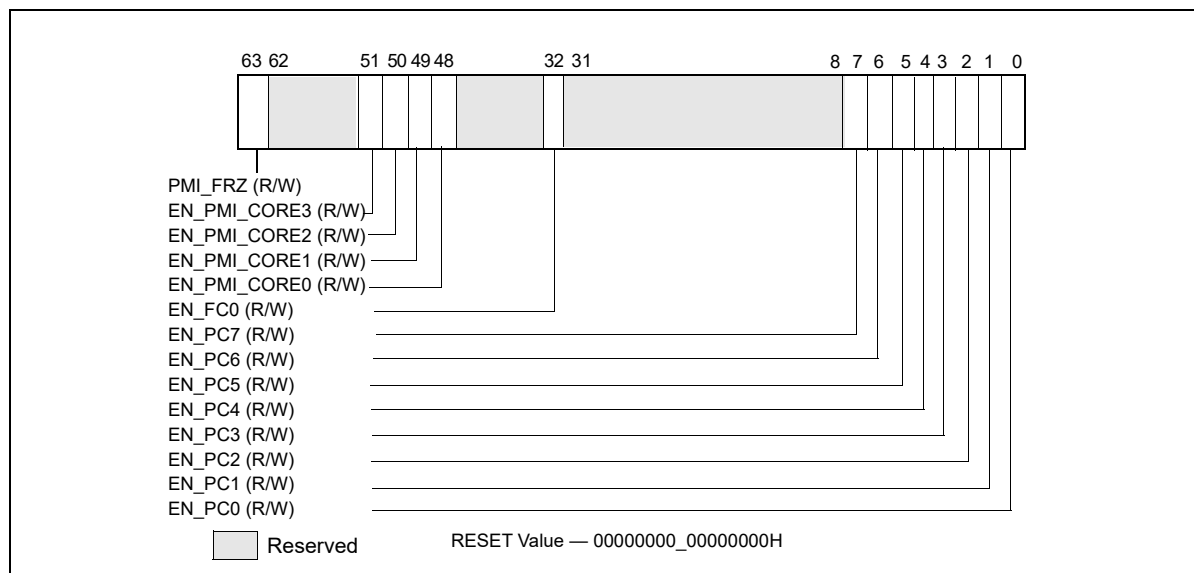


Figure 22-21. Layout of MSR_UNCORE_PERF_GLOBAL_CTRL MSR

MSR_UNCORE_PERF_GLOBAL_STATUS provides overflow status of the U-clock performance counters in the uncore. This is a read-only register. If an overflow status bit is set the corresponding counter has overflowed. The register provides a condition change bit (bit 63) which can be quickly checked by software to determine if a significant change has occurred since the last time the condition change status was cleared. Figure 22-22 shows the layout of MSR_UNCORE_PERF_GLOBAL_STATUS.

- OVF_PcN (bit n, n = 0, 7): When set, indicates general-purpose uncore counter MSR_UNCORE_PerfCntr n has overflowed.
- OVF_FC0 (bit 32): When set, indicates the fixed-function uncore counter MSR_UNCORE_FixedCntr0 has overflowed.
- OVF_PMI (bit 61): When set indicates that an uncore counter overflowed and generated an interrupt request.
- CHG (bit 63): When set indicates that at least one status bit in MSR_UNCORE_PERF_GLOBAL_STATUS register has changed state.

MSR_UNCORE_PERF_GLOBAL_OVF_CTRL allows software to clear the status bits in the UNCORE_PERF_GLOBAL_STATUS register. This is a write-only register, and individual status bits in the global status register are cleared by writing a binary one to the corresponding bit in this register. Writing zero to any bit position in this register has no effect on the uncore PMU hardware.

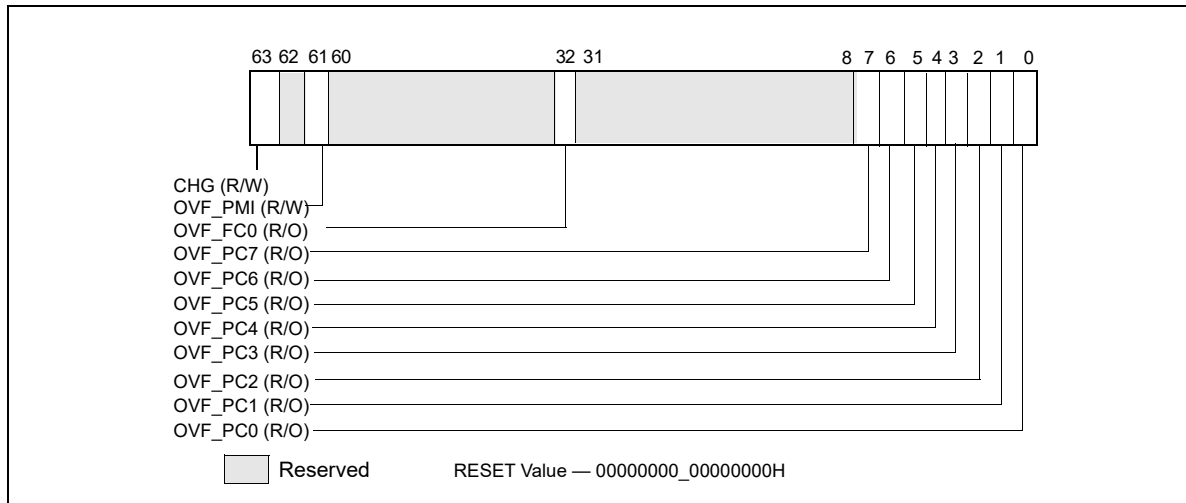


Figure 22-22. Layout of MSR_UNCORE_PERF_GLOBAL_STATUS MSR

Figure 22-23 shows the layout of MSR_UNCORE_PERF_GLOBAL_OVF_CTRL.

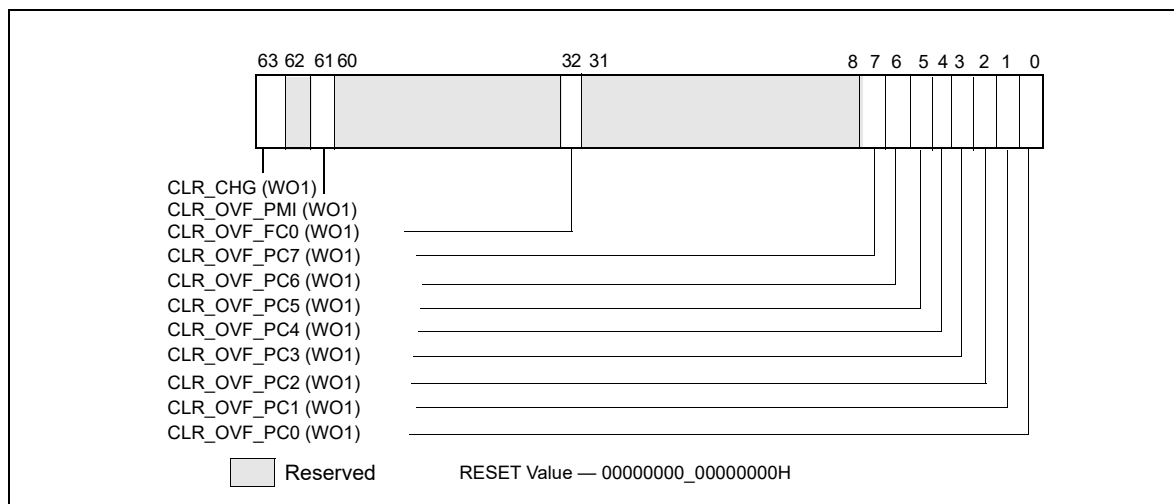


Figure 22-23. Layout of MSR_UNCORE_PERF_GLOBAL_OVF_CTRL MSR

- CLR_OVF_PCn (bit n, n = 0, 7): Set this bit to clear the overflow status for general-purpose uncore counter MSR_UNCORE_PerfCntr n. Writing a value other than 1 is ignored.
- CLR_OVF_FC0 (bit 32): Set this bit to clear the overflow status for the fixed-function uncore counter MSR_UNCORE_FixedCntr0. Writing a value other than 1 is ignored.
- CLR_OVF_PMI (bit 61): Set this bit to clear the OVF_PMI flag in MSR_UNCORE_PERF_GLOBAL_STATUS. Writing a value other than 1 is ignored.
- CLR_CHG (bit 63): Set this bit to clear the CHG flag in MSR_UNCORE_PERF_GLOBAL_STATUS register. Writing a value other than 1 is ignored.

22.3.1.2.2 Uncore Performance Event Configuration Facility

MSR_UNCORE_PerfEvtSel0 through MSR_UNCORE_PerfEvtSel7 are used to select performance event and configure the counting behavior of the respective uncore performance counter. Each uncore PerfEvtSel MSR is paired with an uncore performance counter. Each uncore counter must be locally configured using the corre-

sponding MSR_UNCORE_PerfEvtSelx and counting must be enabled using the respective EN_PCx bit in MSR_UNCORE_PERF_GLOBAL_CTRL. Figure 22-24 shows the layout of MSR_UNCORE_PERFEVTSELx.

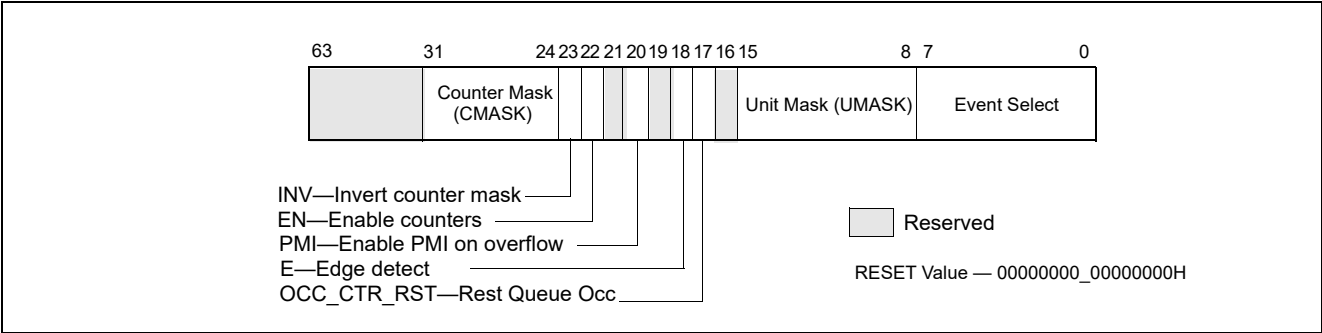


Figure 22-24. Layout of MSR_UNCORE_PERFEVTSELx MSRs

- Event Select (bits 7:0): Selects the event logic unit used to detect uncore events.
- Unit Mask (bits 15:8) : Condition qualifiers for the event selection logic specified in the Event Select field.
- OCC_CTR_RST (bit17): When set causes the queue occupancy counter associated with this event to be cleared (zeroed). Writing a zero to this bit will be ignored. It will always read as a zero.
- Edge Detect (bit 18): When set causes the counter to increment when a deasserted to asserted transition occurs for the conditions that can be expressed by any of the fields in this register.
- PMI (bit 20): When set, the uncore will generate an interrupt request when this counter overflowed. This request will be routed to the logical processors as enabled in the PMI enable bits (EN_PMI_COREx) in the register MSR_UNCORE_PERF_GLOBAL_CTRL.
- EN (bit 22): When clear, this counter is locally disabled. When set, this counter is locally enabled and counting starts when the corresponding EN_PCx bit in MSR_UNCORE_PERF_GLOBAL_CTRL is set.
- INV (bit 23): When clear, the Counter Mask field is interpreted as greater than or equal to. When set, the Counter Mask field is interpreted as less than.
- Counter Mask (bits 31:24): When this field is clear, it has no effect on counting. When set to a value other than zero, the logical processor compares this field to the event counts on each core clock cycle. If INV is clear and the event counts are greater than or equal to this field, the counter is incremented by one. If INV is set and the event counts are less than this field, the counter is incremented by one. Otherwise the counter is not incremented.

Figure 22-25 shows the layout of MSR_UNCORE_FIXED_CTR_CTRL.

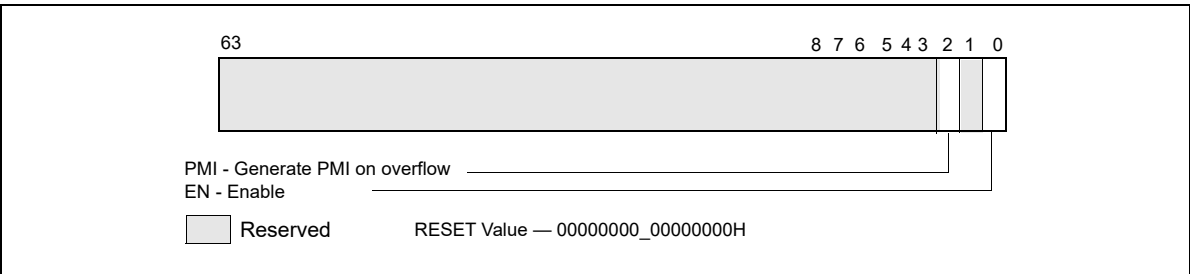


Figure 22-25. Layout of MSR_UNCORE_FIXED_CTR_CTRL MSR

- EN (bit 0): When clear, the uncore fixed-function counter is locally disabled. When set, it is locally enabled and counting starts when the EN_FC0 bit in MSR_UNCORE_PERF_GLOBAL_CTRL is set.
- PMI (bit 2): When set, the uncore will generate an interrupt request when the uncore fixed-function counter overflowed. This request will be routed to the logical processors as enabled in the PMI enable bits (EN_PMI_COREx) in the register MSR_UNCORE_PERF_GLOBAL_CTRL.

Both the general-purpose counters (MSR_UNCORE_PerfCntr) and the fixed-function counter (MSR_UNCORE_FixedCntr0) are 48 bits wide. They support both counting and interrupt based sampling usages. The event logic unit can filter event counts to specific regions of code or transaction types incoming to the home node logic.

22.3.1.2.3 Uncore Address/Opcode Match MSR

The Event Select field [7:0] of MSR_UNCORE_PERFEVTSELx is used to select different uncore event logic unit. When the event "ADDR_OPCODE_MATCH" is selected in the Event Select field, software can filter uncore performance events according to transaction address and certain transaction responses. The address filter and transaction response filtering requires the use of MSR_UNCORE_ADDR_OPCODE_MATCH register. The layout is shown in Figure 22-26.

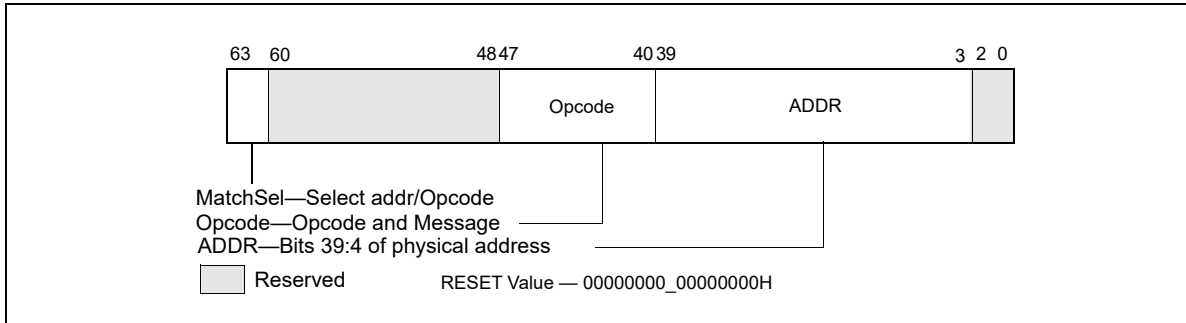


Figure 22-26. Layout of MSR_UNCORE_ADDR_OPCODE_MATCH MSR

- Addr (bits 39:3): The physical address to match if "MatchSel" field is set to select address match. The uncore performance counter will increment if the lowest 40-bit incoming physical address (excluding bits 2:0) for a transaction request matches bits 39:3.
- Opcode (bits 47:40) : Bits 47:40 allow software to filter uncore transactions based on QPI link message class/packed header opcode. These bits are consists two sub-fields:
 - Bits 43:40 specify the QPI packet header opcode.
 - Bits 47:44 specify the QPI message classes.

Table Table 22-8. "Opcode Field Encoding for MSR_UNCORE_ADDR_OPCODE_MATCH" lists the encodings supported in the opcode field.

Table 22-8. Opcode Field Encoding for MSR_UNCORE_ADDR_OPCODE_MATCH

Opcode [43:40]	QPI Message Class		
	Home Request [47:44] = 0000B	Snoop Response [47:44] = 0001B	Data Response [47:44] = 1110B
		1	
DMND_IFETCH	2	2	
WB	3	3	
PF_DATA_RD	4	4	
PF_RFO	5	5	
PF_IFETCH	6	6	
OTHER	7	7	
NON_DRAM	15	15	

- MatchSel (bits 63:61): Software specifies the match criteria according to the following encoding:
 - 000B: Disable addr_opcode match hardware.
 - 100B: Count if only the address field matches.
 - 010B: Count if only the opcode field matches.
 - 110B: Count if either opcode field matches or the address field matches.
 - 001B: Count only if both opcode and address field match.
 - Other encoding are reserved.

22.3.1.3 Intel® Xeon® Processor 7500 Series Performance Monitoring Facility

The performance monitoring facility in the processor core of Intel® Xeon® processor 7500 series are the same as those supported in Intel Xeon processor 5500 series. The uncore subsystem in Intel Xeon processor 7500 series are significantly different The uncore performance monitoring facility consist of many distributed units associated with individual logic control units (referred to as boxes) within the uncore subsystem. A high level block diagram of the various box units of the uncore is shown in Figure 22-27.

Uncore PMUs are programmed via MSR interfaces. Each of the distributed uncore PMU units have several general-purpose counters. Each counter requires an associated event select MSR, and may require additional MSRs to configure sub-event conditions. The uncore PMU MSRs associated with each box can be categorized based on its functional scope: per-counter, per-box, or global across the uncore. The number counters available in each box type are different. Each box generally provides a set of MSRs to enable/disable, check status/overflow of multiple counters within each box.

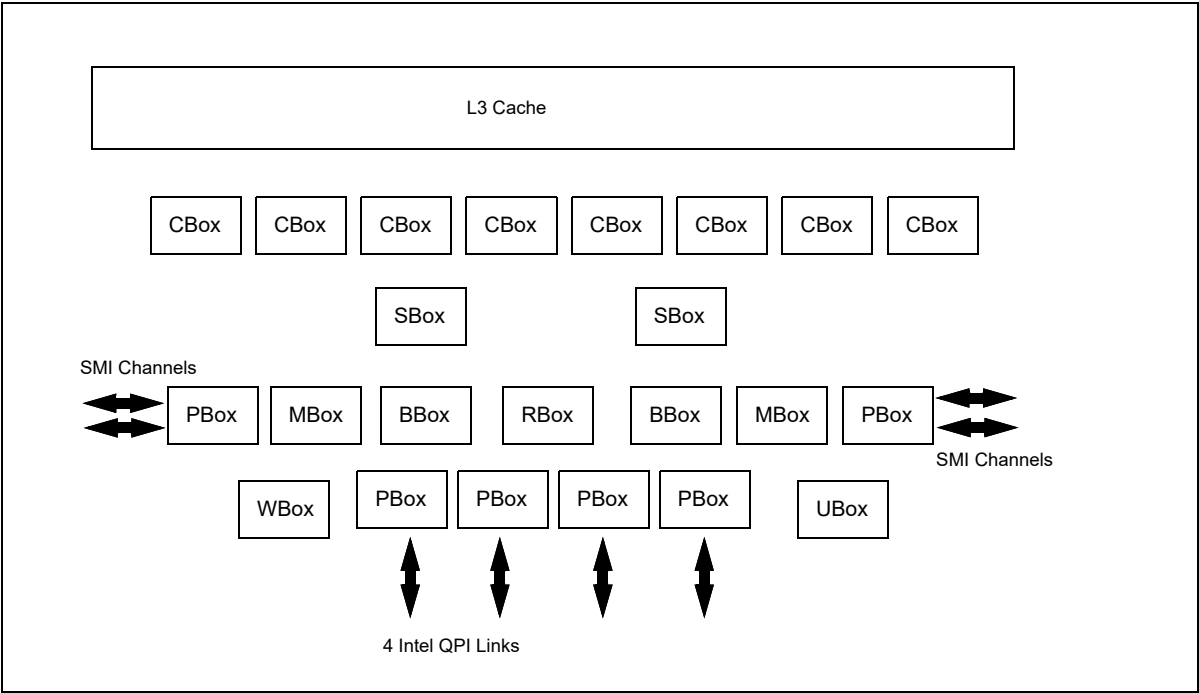


Figure 22-27. Distributed Units of the Uncore of Intel® Xeon® Processor 7500 Series

Table Table 22-9. "Uncore PMU MSR Summary" summarizes the number MSRs for uncore PMU for each box.

Table 22-9. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	6	48	Yes	per-box	None
S-Box	2	4	48	Yes	per-box	Match/Mask
B-Box	2	4	48	Yes	per-box	Match/Mask
M-Box	2	6	48	Yes	per-box	Yes
R-Box	1	16 (2 port, 8 per port)	48	Yes	per-box	Yes
W-Box	1	4	48	Yes	per-box	None
		1	48	No	per-box	None
U-Box	1	1	48	Yes	uncore	None

The W-Box provides 4 general-purpose counters, each requiring an event select configuration MSR, similar to the general-purpose counters in other boxes. There is also a fixed-function counter that increments clockticks in the uncore clock domain.

For C,S,B,M,R, and W boxes, each box provides an MSR to enable/disable counting, configuring PMI of multiple counters within the same box, this is somewhat similar to the “global control” programming interface, IA32_PERF_GLOBAL_CTRL, offered in the core PMU. Similarly status information and counter overflow control for multiple counters within the same box are also provided in C,S,B,M,R, and W boxes.

In the U-Box, MSR_U_PMON_GLOBAL_CTL provides overall uncore PMU enable/disable and PMI configuration control. The scope of status information in the U-box is at per-box granularity, in contrast to the per-box status information MSR (in the C,S,B,M,R, and W boxes) providing status information of individual counter overflow. The difference in scope also apply to the overflow control MSR in the U-Box versus those in the other Boxes.

The individual MSRs that provide uncore PMU interfaces are listed in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, Table 2-17. “Additional MSRs in the Intel® Xeon® Processor 7500 Series” under the general naming style of MSR_%box#%_PMON_%scope_function%, where %box#% designates the type of box and zero-based index if there are more than one box of the same type, %scope_function% follows the examples below:

- Multi-counter enabling MSRs: MSR_U_PMON_GLOBAL_CTL, MSR_S0_PMON_BOX_CTL, MSR_C7_PMON_BOX_CTL, etc.
- Multi-counter status MSRs: MSR_U_PMON_GLOBAL_STATUS, MSR_S0_PMON_BOX_STATUS, MSR_C7_PMON_BOX_STATUS, etc.
- Multi-counter overflow control MSRs: MSR_U_PMON_GLOBAL_OVF_CTL, MSR_S0_PMON_BOX_OVF_CTL, MSR_C7_PMON_BOX_OVF_CTL, etc.
- Performance counters MSRs: the scope is implicitly per counter, e.g., MSR_U_PMON_CTR, MSR_S0_PMON_CTR0, MSR_C7_PMON_CTR5, etc.
- Event select MSRs: the scope is implicitly per counter, e.g., MSR_U_PMON_EVNT_SEL, MSR_S0_PMON_EVNT_SEL0, MSR_C7_PMON_EVNT_SEL5, etc.
- Sub-control MSRs: the scope is implicitly per-box granularity, e.g., MSR_M0_PMON_TIMESTAMP, MSR_R0_PMON_IPERF0_P1, MSR_S1_PMON_MATCH.

Details of uncore PMU MSR bit field definitions can be found in a separate document “Intel Xeon Processor 7500 Series Uncore Performance Monitoring Guide”.

22.3.2 Performance Monitoring for Processors Based on Westmere Microarchitecture

All of the performance monitoring programming interfaces (architectural and non-architectural core PMU facilities, and uncore PMU) described in Section 22.6.3 also apply to processors based on Westmere microarchitecture.

Table Table 22-6. "Off-Core Response Event Encoding" describes a non-architectural performance monitoring event (event code 0B7H) and associated MSR_OFFCORE_RSP_0 (address 1A6H) in the core PMU. This event and a second functionally equivalent offcore response event using event code 0BBH and MSR_OFFCORE_RSP_1 (address 1A7H) are supported in processors based on Westmere microarchitecture. The event code and event mask definitions of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

The load latency facility is the same as described in Section 22.3.1.1.2, but added enhancement to provide more information in the data source encoding field of each load latency record. The additional information relates to STLB_MISS and LOCK, see Table Table 22-14. "Layout of Data Source Field of Load Latency Record".

22.3.3 Intel® Xeon® Processor E7 Family Performance Monitoring Facility

The performance monitoring facility in the processor core of the Intel® Xeon® processor E7 family is the same as those supported in the Intel Xeon processor 5600 series¹. The uncore subsystem in the Intel Xeon processor E7 family is similar to those of the Intel Xeon processor 7500 series. The high level construction of the uncore subsystem is similar to that shown in Figure 22-27, with the additional capability that up to 10 C-Box units are supported.

Table Table 22-10. "Uncore PMU MSR Summary for Intel® Xeon® Processor E7 Family" summarizes the number MSRs for uncore PMU for each box.

Table 22-10. Uncore PMU MSR Summary for Intel® Xeon® Processor E7 Family

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	10	6	48	Yes	per-box	None
S-Box	2	4	48	Yes	per-box	Match/Mask
B-Box	2	4	48	Yes	per-box	Match/Mask
M-Box	2	6	48	Yes	per-box	Yes
R-Box	1	16 (2 port, 8 per port)	48	Yes	per-box	Yes
W-Box	1	4	48	Yes	per-box	None
		1	48	No	per-box	None
U-Box	1	1	48	Yes	uncore	None

Details of the uncore performance monitoring facility of Intel Xeon Processor E7 family is available in the "Intel® Xeon® Processor E7 Uncore Performance Monitoring Programming Reference Manual".

22.3.4 Performance Monitoring for Processors Based on Sandy Bridge Microarchitecture

Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, and Intel® Xeon® processor E3-1200 family are based on Sandy Bridge microarchitecture; this section describes the performance monitoring facilities provided in the processor core. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 22.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 22.2.3.

The core PMU's capability is similar to those described in Section 22.3.1.1 and Section 22.6.3, with some differences and enhancements relative to Westmere microarchitecture summarized in Table Table 22-11. "Core PMU Comparison".

1. Exceptions are indicated for event code 0FH in the event list for this processor (<https://perfmon-events.intel.com/>); and valid bits of data source encoding field of each load latency record is limited to bits 5:4 of Table Table 22-14. "Layout of Data Source Field of Load Latency Record".

Table 22-11. Core PMU Comparison

Box	Sandy Bridge Microarchitecture	Westmere Microarchitecture	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W:32	See Section 22.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4	Use CPUID to determine # of counters. See Section 22.2.1.
PMI Overhead Mitigation	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling. - Freeze_while_SMM.	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling. - Freeze_while_SMM.	See Section 20.4.7.
Processor Event Based Sampling (PEBS) Events	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	See Table Table 22-88. "PEBS Performance Events for Intel Core Microarchitecture".	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 22.3.4.4.2; - Data source encoding - STLB miss encoding - Lock transaction encoding	Data source encoding	
PEBS-Precise Store	Section 22.3.4.4.3	No	
PEBS-PDIR	Yes (using precise INST_RETIRED.ALL).	No	
Off-core Response Event	MSR 1A6H and 1A7H, extended request and response types.	MSR 1A6H and 1A7H, limited response types.	Nehalem supports 1A6H only.

22.3.4.1 Global Counter Control Facilities in Sandy Bridge Microarchitecture

The number of general-purpose performance counters visible to a logical processor can vary across Processors based on Sandy Bridge microarchitecture. Software must use CPUID to determine the number performance counters/event select registers (See Section).

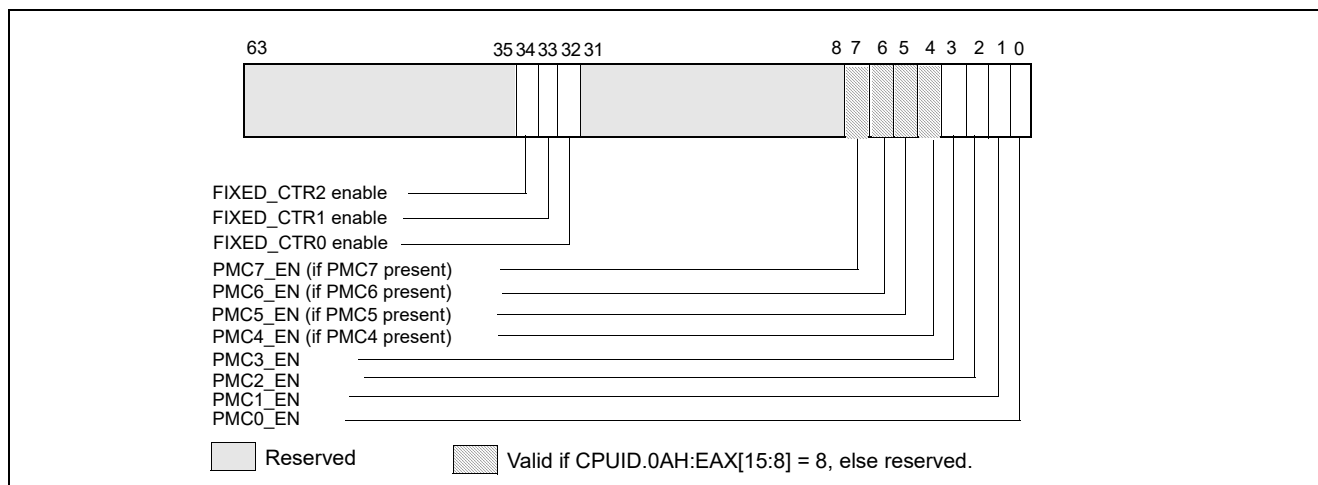


Figure 22-28. IA32_PERF_GLOBAL_CTRL MSR in Sandy Bridge Microarchitecture

Figure 22-46 depicts the layout of IA32_PERF_GLOBAL_CTRL MSR. The enable bits (PMC4_EN, PMC5_EN, PMC6_EN, PMC7_EN) corresponding to IA32_PMC4-IA32_PMC7 are valid only if CPUID.0AH:EAX[15:8] reports a value of '8'. If CPUID.0AH:EAX[15:8] = 4, attempts to set the invalid bits will cause #GP.

Each enable bit in IA32_PERF_GLOBAL_CTRL is ANDed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_PERF_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the ANDed results is true; counting is disabled when the result is false. IA32_PERF_GLOBAL_STATUS MSR provides single-bit status used by software to query the overflow condition of each performance counter. IA32_PERF_GLOBAL_STATUS[bit 62] indicates overflow conditions of the DS area data buffer (see Figure 22-29). A value of 1 in each bit of the PMCx_OVF field indicates an overflow condition has occurred in the associated counter.

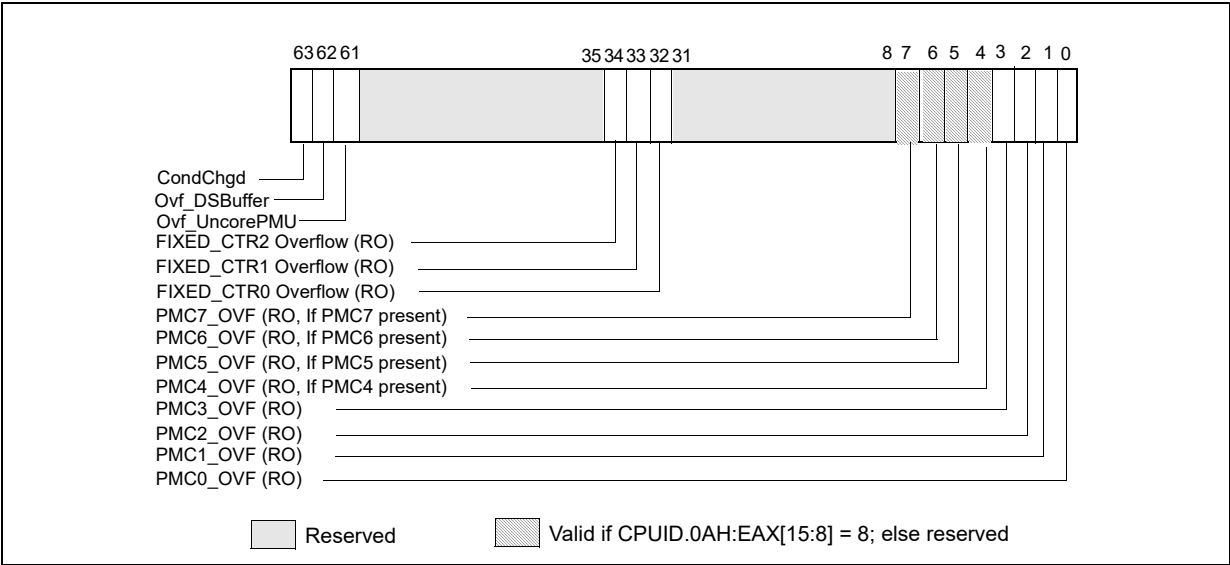


Figure 22-29. IA32_PERF_GLOBAL_STATUS MSR in Sandy Bridge Microarchitecture

When a performance counter is configured for PEBS, an overflow condition in the counter will arm PEBS. On the subsequent event following overflow, the processor will generate a PEBS event. On a PEBS event, the processor will perform bounds checks based on the parameters defined in the DS Save Area (see Section 20.4.9). Upon successful bounds checks, the processor will store the data record in the defined buffer area, clear the counter overflow status, and reload the counter. If the bounds checks fail, the PEBS will be skipped entirely. In the event that the PEBS buffer fills up, the processor will set the OvfBuffer bit in MSR_PERF_GLOBAL_STATUS.

IA32_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow the indicators for general-purpose or fixed-function counters via a single WRMSR (see Figure 22-30). Clear overflow indications when:

- Setting up new values in the event select and/or UMASK field for counting or interrupt based sampling.
- Reloading counter values to continue sampling.
- Disabling event counting or interrupt based sampling.

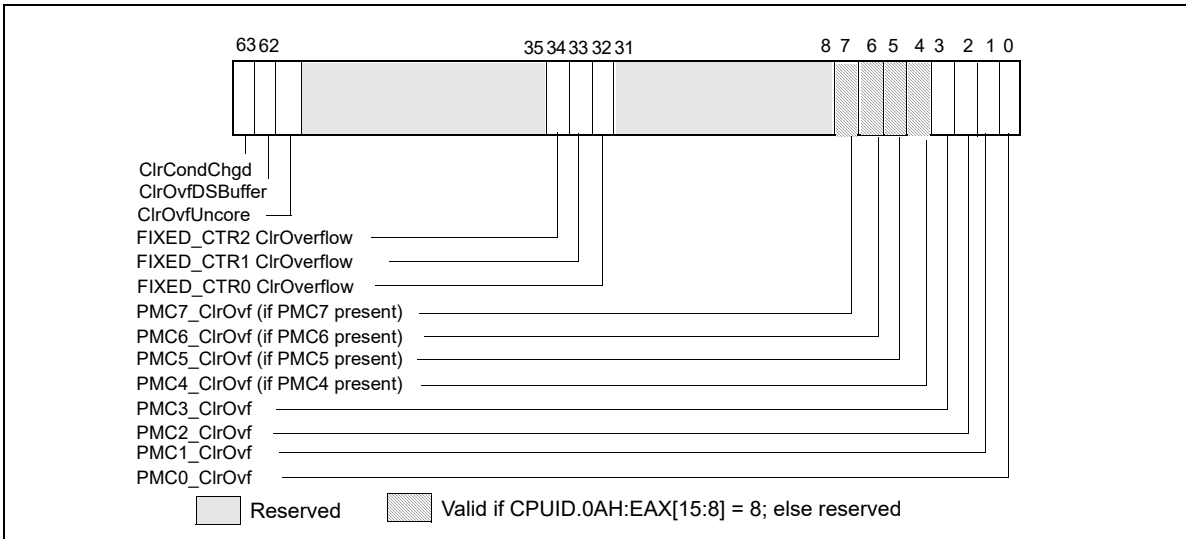


Figure 22-30. IA32_PERF_GLOBAL_OVF_CTRL MSR in Sandy Bridge Microarchitecture

22.3.4.2 Counter Coalescence

In processors based on Sandy Bridge microarchitecture, each processor core implements eight general-purpose counters. CPUID.0AH:EAX[15:8] will report the number of counters visible to software.

If a processor core is shared by two logical processors, each logical processors can access up to four counters (IA32_PMC0-IA32_PMC3). This is the same as in the prior generation for processors based on Nehalem microarchitecture.

If a processor core is not shared by two logical processors, up to eight general-purpose counters are visible. If CPUID.0AH:EAX[15:8] reports 8 counters, then IA32_PMC4-IA32_PMC7 would occupy MSR addresses 0C5H through 0C8H. Each counter is accompanied by an event select MSR (IA32_PERFEVTSEL4-IA32_PERFEVTSEL7).

If CPUID.0AH:EAX[15:8] report 4, access to IA32_PMC4-IA32_PMC7, IA32_PMC4-IA32_PMC7 will cause #GP. Writing 1's to bit position 7:4 of IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, or IA32_PERF_GLOBAL_OVF_CTL will also cause #GP.

22.3.4.3 Full Width Writes to Performance Counters

Processors based on Sandy Bridge microarchitecture support full-width writes to the general-purpose counters, IA32_PMCx. Support of full-width writes are enumerated by IA32_PERF_CAPABILITIES.FW_WRITES[13] (see Section 22.2.4).

The default behavior of IA32_PMCx is unchanged, i.e., WRMSR to IA32_PMCx results in a sign-extended 32-bit value of the input EAX written into IA32_PMCx. Full-width writes must issue WRMSR to a dedicated alias MSR address for each IA32_PMCx.

Software must check the presence of full-width write capability and the presence of the alias address IA32_A_PMCx by testing IA32_PERF_CAPABILITIES[13].

22.3.4.4 PEBS Support in Sandy Bridge Microarchitecture

Processors based on Sandy Bridge microarchitecture support PEBS, similar to those offered in prior generation, with several enhanced features. The key components and differences of PEBS facility relative to Westmere microarchitecture is summarized in Table Table 22-12. "PEBS Facility Comparison".

Table 22-12. PEBS Facility Comparison

Box	Sandy Bridge Microarchitecture	Westmere Microarchitecture	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7.
PEBS Buffer Programming	Section 22.3.1.1.1	Section 22.3.1.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 22-31	Figure 22-17	
PEBS record layout	Physical Layout same as Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family".	Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family"	Enhanced fields at offsets 98H, A0H, A8H.
PEBS Events	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	See Table Table 22-88. "PEBS Performance Events for Intel Core Microarchitecture".	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table Table 22-14. "Layout of Data Source Field of Load Latency Record".	Table Table 22-5. "Data Source Encoding for Load Latency Record"	
PEBS-Precise Store	Yes; see Section 22.3.4.4.3.	No	IA32_PMC3 only
PEBS-PDIR	Yes	No	IA32_PMC1 only
PEBS skid from EventingIP	1 (or 2 if micro+macro fusion)	1	
SAMPLING Restriction	Small SAV(CountDown) value incur higher overhead than prior generation.		

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

In IA32_PEBS_ENABLE MSR, bit 63 is defined as PS_ENABLE: When set, this enables IA32_PMC3 to capture precise store information. Only IA32_PMC3 supports the precise store facility. In typical usage of PEBS, the bit fields in IA32_PEBS_ENABLE are written to when the agent software starts PEBS operation; the enabled bit fields should be modified only when re-programming another PEBS event or cleared when the agent uses the performance counters for non-PEBS operations.

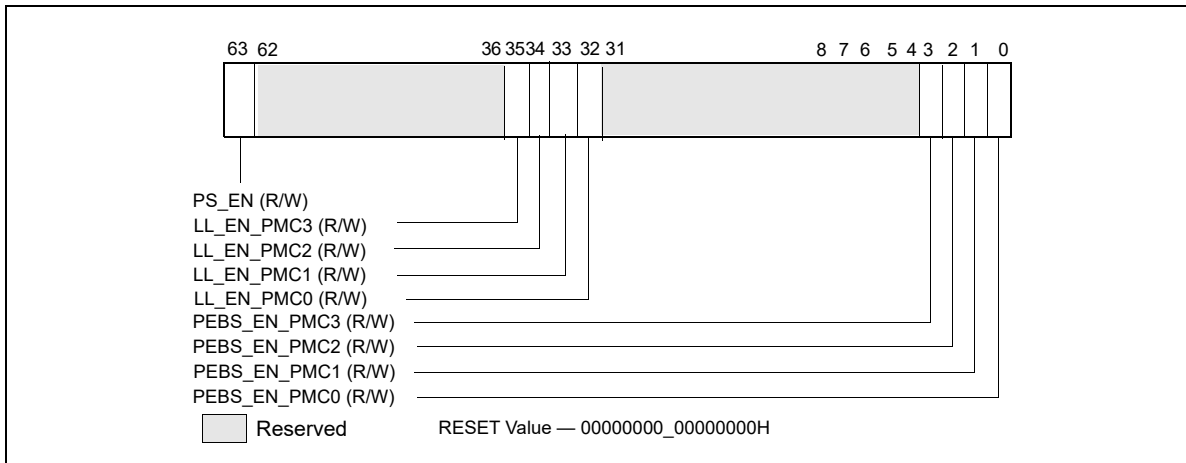


Figure 22-31. Layout of IA32_PEBS_ENABLE MSR

22.3.4.4.1 PEBS Record Format

The layout of PEBS records physically identical to those shown in Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family", but the fields at offsets 98H, A0H, and A8H have been enhanced to support additional PEBS capabilities.

- Load/Store Data Linear Address (Offset 98H): This field will contain the linear address of the source of the load, or linear address of the destination of the store.
- Data Source /Store Status (Offset A0H): When load latency is enabled, this field will contain three piece of information (including an encoded value indicating the source which satisfied the load operation). The source field encodings are detailed in Table Table 22-5. "Data Source Encoding for Load Latency Record". When precise store is enabled, this field will contain information indicating the status of the store, as detailed in Table 19.
- Latency Value/0 (Offset A8H): When load latency is enabled, this field contains the latency in cycles to service the load. This field is not meaningful when precise store is enabled and will be written to zero in that case. Upon writing the PEBS record, microcode clears the overflow status bits in the IA32_PERF_GLOBAL_STATUS corresponding to those counters that both overflowed and were enabled in the IA32_PEBS_ENABLE register. The status bits of other counters remain unaffected.

The number PEBS events has expanded. The list of PEBS events supported in Sandy Bridge microarchitecture is shown in Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".

Table 22-13. PEBS Performance Events for Sandy Bridge Microarchitecture

Event Name	Event Select	Sub-event	UMask
INST_RETIRED	C0H	PREC_DIST	01H ¹
UOPS_RETIRED	C2H	All	01H
		Retire_Slots	02H
BR_INST_RETIRED	C4H	Conditional	01H
		Near_Call	02H
		All_branches	04H
		Near_Return	08H
		Near_Taken	20H

Table 22-13. PEBS Performance Events for Sandy Bridge Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
BR_MISP_RETIRED	C5H	Conditional	01H
		Near_Call	02H
		All_branches	04H
		Not_Taken	10H
		Taken	20H
MEM_UOPS_RETIRED	D0H	STLB_MISS_LOADS	11H
		STLB_MISS_STORE	12H
		LOCK_LOADS	21H
		SPLIT_LOADS	41H
		SPLIT_STORES	42H
		ALL_LOADS	81H
		ALL_STORES	82H
MEM_LOAD_UOPS_RETIRED	D1H	L1_Hit	01H
		L2_Hit	02H
		L3_Hit	04H
		Hit_LFB	40H
MEM_LOAD_UOPS_LLC_HIT_RETIRED	D2H	XSNP_Miss	01H
		XSNP_Hit	02H
		XSNP_Hitm	04H
		XSNP_None	08H

NOTES:

1. Only available on IA32_PMC1.

22.3.4.4.2 Load Latency Performance Monitoring Facility

The load latency facility in Sandy Bridge microarchitecture is similar to that in prior microarchitectures. It provides software a means to characterize the average load latency to different levels of cache/memory hierarchy. This facility requires processor supporting enhanced PEBS record format in the PEBS buffer, see Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family" and Section 22.3.4.4.1. This field measures the load latency from load's first dispatch of till final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches).

To use this feature software must assure:

- One of the IA32_PERFEVTSELx MSR is programmed to specify the event unit MEM_TRANS_RETIRED, and the LATENCY_ABOVE_THRESHOLD event mask must be specified (IA32_PerfEvtSelX[15:0] = 1CDH). The corresponding counter IA32_PMCx will accumulate event counts for architecturally visible loads which exceed the programmed latency threshold specified separately in a MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.
- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with latencies greater than this value are eligible for counting and latency data reporting. The minimum value that may be programmed in this register is 3 (the minimum detectable load latency is 4 core clock cycles).
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register. This means that both the PEBS_EN_CTRX and LL_EN_CTRX bits must be set for the counter(s) of interest. For example, to enable load latency on counter IA32_PMC0, the IA32_PEBS_ENABLE register must be programmed with the 64-bit value 00000001.00000001H.

- When Load latency event is enabled, no other PEBS event can be configured with other counters.

When the load-latency facility is enabled, load operations are randomly selected by hardware and tagged to carry information related to data source locality and latency. Latency and data source information of tagged loads are updated internally. The MEM_TRANS_RETIRE event for load latency counts only tagged retired loads. If a load is cancelled it will not be counted and the internal state of the load latency facility will not be updated. In this case the hardware will tag the next available load.

When a PEBS assist occurs, the last update of latency and data source information are captured by the assist and written as part of the PEBS record. The PEBS sample after value (SAV), specified in PEBS CounterX Reset, operates orthogonally to the tagging mechanism. Loads are randomly tagged to collect latency data. The SAV controls the number of tagged loads with latency information that will be written into the PEBS record field by the PEBS assists. The load latency data written to the PEBS record will be for the last tagged load operation which retired just before the PEBS assist was invoked.

The physical layout of the PEBS records is the same as shown in Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family". The specificity of Data Source entry at offset A0H has been enhanced to report three pieces of information.

Table 22-14. Layout of Data Source Field of Load Latency Record

Field	Position	Description
Source	3:0	See Table Table 22-5. "Data Source Encoding for Load Latency Record"
STLB_MISS	4	0: The load did not miss the STLB (hit the DTLB or STLB). 1: The load missed the STLB.
Lock	5	0: The load was not part of a locked transaction. 1: The load was part of a locked transaction.
Reserved	63:6	Reserved

The layout of MSR_PEBS_LD_LAT_THRESHOLD is the same as shown in Figure 22-19.

22.3.4.4.3 Precise Store Facility

Processors based on Sandy Bridge microarchitecture offer a precise store capability that complements the load latency facility. It provides a means to profile store memory references in the system.

Precise stores leverage the PEBS facility and provide additional information about sampled stores. Having precise memory reference events with linear address information for both loads and stores can help programmers improve data structure layout, eliminate remote node references, and identify cache-line conflicts in NUMA systems.

Only IA32_PMC3 can be used to capture precise store information. After enabling this facility, counter overflows will initiate the generation of PEBS records as previously described in PEBS. Upon counter overflow hardware captures the linear address and other status information of the next store that retires. This information is then written to the PEBS record.

To enable the precise store facility, software must complete the following steps. Please note that the precise store facility relies on the PEBS facility, so the PEBS configuration requirements must be completed before attempting to capture precise store information.

- Complete the PEBS configuration steps.
- Program the MEM_TRANS_RETIRED.PRECISE_STORE event in IA32_PERFEVTSEL3. Only counter 3 (IA32_PMC3) supports collection of precise store information.
- Set IA32_PEBS_ENABLE[3] and IA32_PEBS_ENABLE[63]. This enables IA32_PMC3 as a PEBS counter and enables the precise store facility, respectively.

The precise store information written into a PEBS record affects entries at offsets 98H, A0H, and A8H of Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family". The specificity of Data Source entry at offset A0H has been enhanced to report three piece of information.

Table 22-15. Layout of Precise Store Information In PEBS Record

Field	Offset	Description
Store Data Linear Address	98H	The linear address of the destination of the store.
Store Status	A0H	L1D Hit (Bit 0): The store hit the data cache closest to the core (lowest latency cache) if this bit is set, otherwise the store missed the data cache. STLB Miss (bit 4): The store missed the STLB if set, otherwise the store hit the STLB Locked Access (bit 5): The store was part of a locked access if set, otherwise the store was not part of a locked access.
Reserved	A8H	Reserved

22.3.4.4.4 Precise Distribution of Instructions Retired (PDIR)

Upon triggering a PEBS assist, there will be a finite delay between the time the counter overflows and when the microcode starts to carry out its data collection obligations. INST_RETIRE is a very common event that is used to sample where performance bottleneck happened and to help identify its location in instruction address space. Even if the delay is constant in core clock space, it invariably manifest as variable “skids” in instruction address space. This creates a challenge for programmers to profile a workload and pinpoint the location of bottlenecks.

The core PMU in processors based on Sandy Bridge microarchitecture include a facility referred to as precise distribution of Instruction Retired (PDIR).

The PDIR facility mitigates the “skid” problem by providing an early indication of when the INST_RETIRE counter is about to overflow, allowing the machine to more precisely trap on the instruction that actually caused the counter overflow. On processors based on Sandy Bridge microarchitecture, skid is significantly reduced and can be as little as one instruction. On future implementations, PDIR may eliminate skid.

PDIR applies only to the INST_RETIRE.ALL precise event, and processors based on Sandy Bridge microarchitecture must use IA32_PMC1 with PerfEvtSel1 property configured and bit 1 in the IA32_PEBS_ENABLE set to 1. INST_RETIRE.ALL is a non-architectural performance event, it is not supported in prior generation microarchitectures. Additionally, on processors with CPUID DisplayFamily_DisplayModel signatures of 06_2A and 06_2D, the tool that programs PDIR should quiesce the rest of the programmable counters in the core when PDIR is active.

22.3.4.5 Off-core Response Performance Monitoring

The core PMU in processors based on Sandy Bridge microarchitecture provides off-core response facility similar to prior generation. Off-core response can be programmed only with a specific pair of event select and counter MSR, and with specific event codes and predefine mask bit value in a dedicated MSR to specify attributes of the off-core transaction. Two event codes are dedicated for off-core response event programming. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Table 22-16 lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 22-16. Off-Core Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMC0-3	B7H	01H	MSR_OFFCORE_RSP_0 (address 1A6H)
PMC0-3	BBH	01H	MSR_OFFCORE_RSP_1 (address 1A7H)

The layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 are shown in Figure 22-32 and Figure 22-33. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

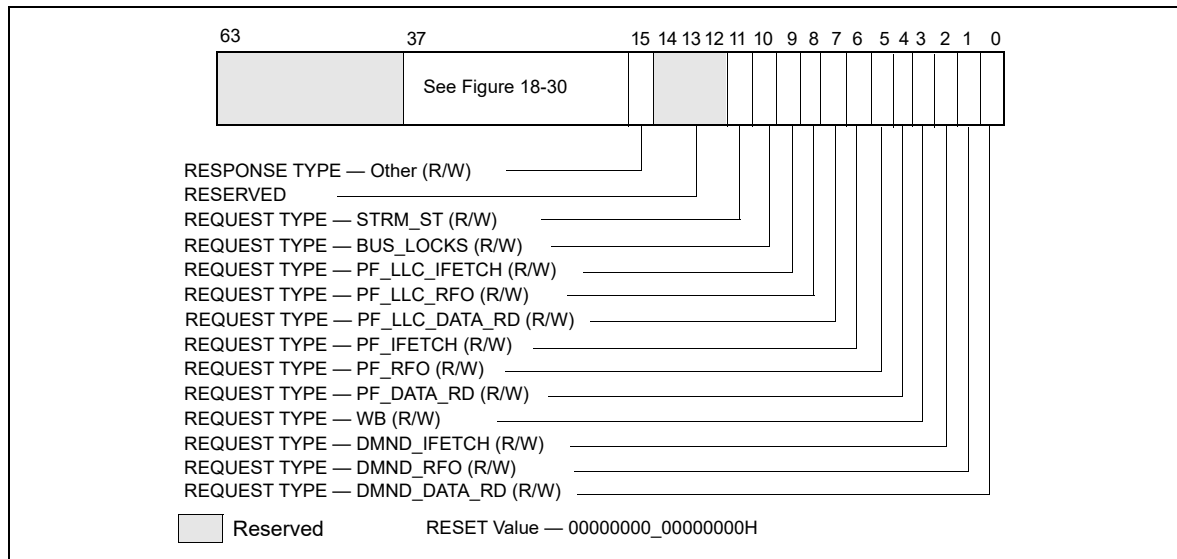


Figure 22-32. Request_Type Fields for MSR_OFFCORE_RSP_x

Table 22-17. MSR_OFFCORE_RSP_x Request_Type Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PF_LLC_DATA_RD	7	L2 prefetcher to L3 for loads.
PF_LLC_RFO	8	RFO requests generated by L2 prefetcher
PF_LLC_IFETCH	9	L2 prefetcher to L3 for instruction fetches.
BUS_LOCKS	10	Bus lock and split lock requests
STRM_ST	11	Streaming store requests
OTHER	15	Any other request that crosses IDI, including I/O.

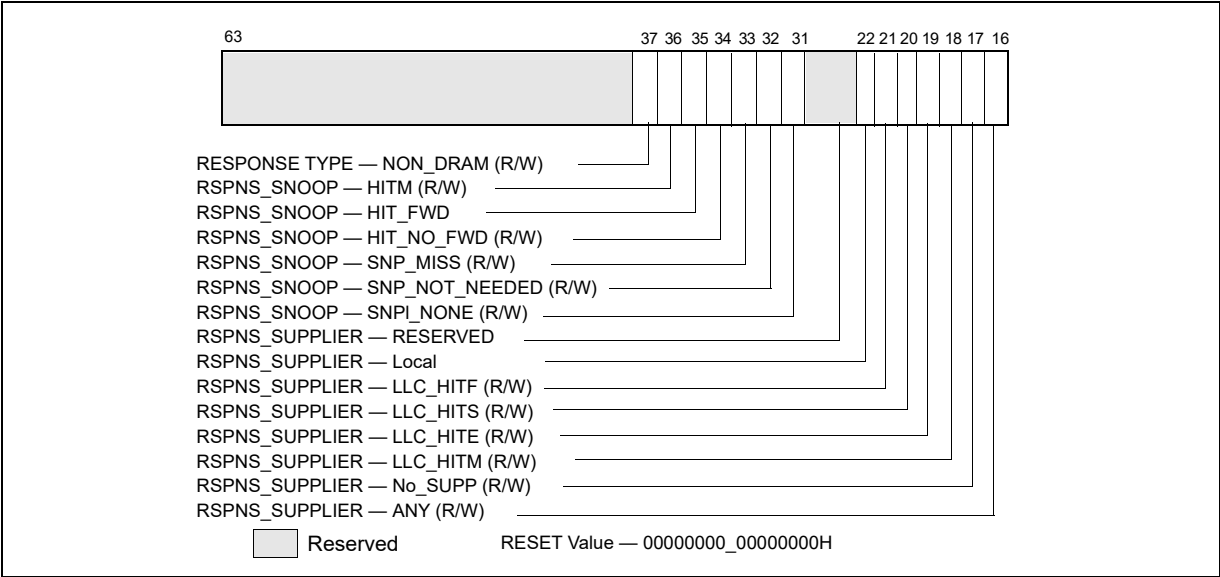


Figure 22-33. Response_Supplier and Snoop Info Fields for MSR_OFFCORE_RSP_x

To properly program this extra register, software must set at least one request type bit and a valid response type pattern. Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSP_x allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 22-18. MSR_OFFCORE_RSP_x Response Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	LLC_HITM	18	M-state initial lookup stat in L3.
	LLC_HITE	19	E-state
	LLC_HITS	20	S-state
	LLC_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Reserved	30:23	Reserved

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

ANY | [(‘OR’ of Supplier Info Bits) & (‘OR’ of Snoop Info Bits)]

If “ANY” bit is set, the supplier and snoop info bits are ignored.

Table 22-19. MSR_OFFCORE_RSP_x Snoop Info Field Definition

Subtype	Bit Name	Offset	Description
Snoop Info	SNP_NONE	31	No details on snoop-related information.
	SNP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNP_MISS	33	A snoop was needed and it missed all snooped caches: -For LLC Hit, ReslHitl was returned by all cores -For LLC Miss, Rspl was returned by all sockets and data was returned from DRAM.
	SNP_NO_FWD	34	A snoop was needed and it hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. This includes: -Snoop Hit w/ Invalidation (LLC Hit, RFO) -Snoop Hit, Left Shared (LLC Hit/Miss, IFetch/Data_RD) -Snoop Hit w/ Invalidation and No Forward (LLC Miss, RFO Hit S) In the LLC Miss case, data is returned from DRAM.
	SNP_FWD	35	A snoop was needed and data was forwarded from a remote socket. This includes: -Snoop Forward Clean, Left Shared (LLC Hit/Miss, IFetch/Data_RD/RFT).
	HITM	36	A snoop was needed and it HitM-ed in local or remote cache. HitM denotes a cache-line was in modified state before effect as a results of snoop. This includes: -Snoop HitM w/ WB (LLC miss, IFetch/Data_RD) -Snoop Forward Modified w/ Invalidation (LLC Hit/Miss, RFO) -Snoop MtoS (LLC Hit, IFetch/Data_RD).
	NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.

22.3.4.6 Uncore Performance Monitoring Facilities in the Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, and Intel® Core™ i3-2xxx Processor Series

The uncore sub-system in Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series provides a unified L3 that can support up to four processor cores. The L3 cache consists multiple slices, each slice interface with a processor via a coherence engine, referred to as a C-Box. Each C-Box provides dedicated facility of MSRs to select uncore performance monitoring events and each C-Box event select MSR is paired with a counter register, similar in style as those described in Section 22.3.1.2.2. The ARB unit in the uncore also provides its local performance counters and event select MSRs. The layout of the event select MSRs in the C-Boxes and the ARB unit are shown in Figure 22-34.

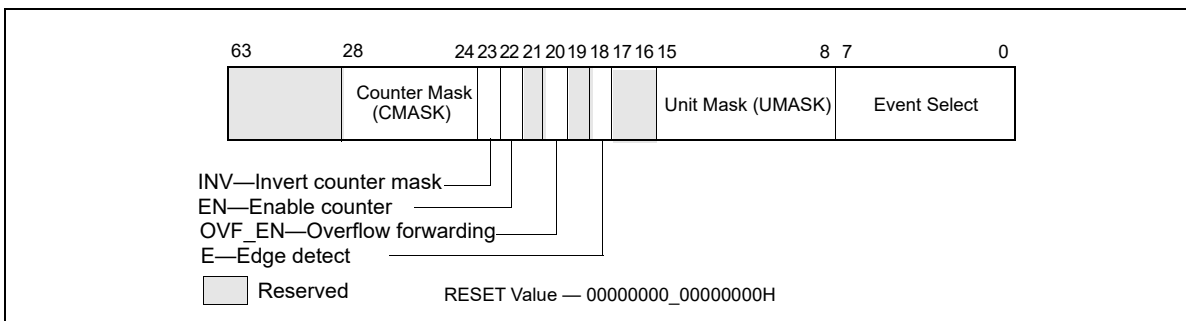


Figure 22-34. Layout of Uncore PERFECTSEL MSR for a C-Box Unit or the ARB Unit

The bit fields of the uncore event select MSRs for a C-box unit or the ARB unit are summarized below:

- Event_Select (bits 7:0) and UMASK (bits 15:8): Specifies the microarchitectural condition to count in a local uncore PMU counter, see the event list at: <https://perfmon-events.intel.com/>.
- E (bit 18): Enables edge detection filtering, if 1.
- OVF_EN (bit 20): Enables the overflow indicator from the uncore counter forwarded to MSR_UNC_PERF_-GLOBAL_CTRL, if 1.
- EN (bit 22): Enables the local counter associated with this event select MSR.
- INV (bit 23): Event count increments with non-negative value if 0, with negated value if 1.
- CMASK (bits 28:24): Specifies a positive threshold value to filter raw event count input.

At the uncore domain level, there is a master set of control MSRs that centrally manages all the performance monitoring facility of uncore units. Figure 22-35 shows the layout of the uncore domain global control.

When an uncore counter overflows, a PMI can be routed to a processor core. Bits 3:0 of MSR_UNC_PERF_-GLOBAL_CTRL can be used to select which processor core to handle the uncore PMI. Software must then write to bit 13 of IA32_DEBUGCTL (at address 1D9H) to enable this capability.

- PMI_SEL_Core#: Enables the forwarding of an uncore PMI request to a processor core, if 1. If bit 30 (WakePMI) is '1', a wake request is sent to the respective processor core prior to sending the PMI.
- EN: Enables the fixed uncore counter, the ARB counters, and the CBO counters in the uncore PMU, if 1. This bit is cleared if bit 31 (FREEZE) is set and any enabled uncore counters overflow.
- WakePMI: Controls sending a wake request to any halted processor core before issuing the uncore PMI request. If a processor core was halted and not sent a wake request, the uncore PMI will not be serviced by the processor core.
- FREEZE: Provides the capability to freeze all uncore counters when an overflow condition occurs in a unit counter. When this bit is set, and a counter overflow occurs, the uncore PMU logic will clear the global enable bit (bit 29).

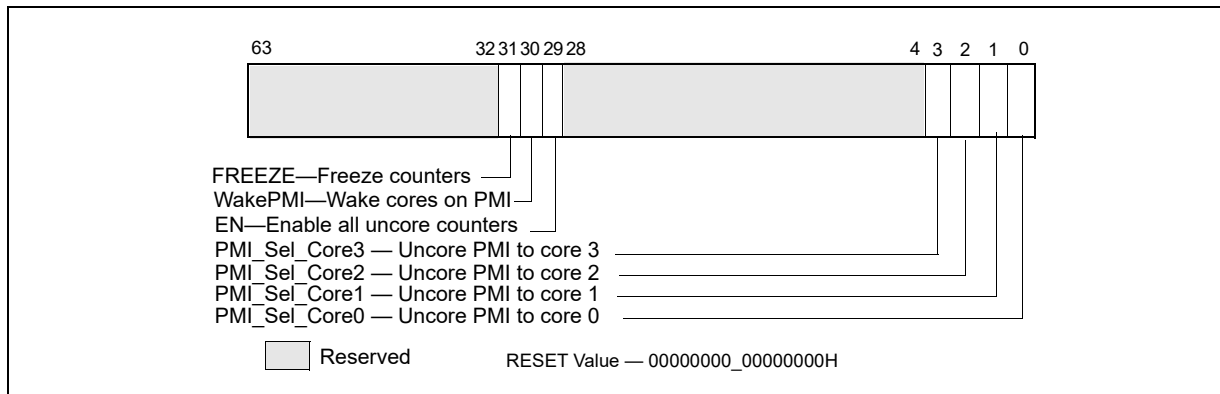


Figure 22-35. Layout of MSR_UNC_PERF_GLOBAL_CTRL MSR for Uncore

Additionally, there is also a fixed counter, counting uncore clockticks, for the uncore domain. Table Table 22-20. "Uncore PMU MSR Summary" summarizes the number MSRs for uncore PMU for each box.

Table 22-20. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
C-Box	SKU specific	2	44	Yes	Per-box	Up to 4, seeTable Table 2-21. "MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)" MSR_UNC_CBO_CONFIG

Table 22-20. Uncore PMU MSR Summary (Contd.)

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
ARB	1	2	44	Yes	Uncore	
Fixed Counter	N.A.	N.A.	48	No	Uncore	

22.3.4.6.1 Uncore Performance Monitoring Events

There are certain restrictions on the uncore performance counters in each C-Box. Specifically,

- Occupancy events are supported only with counter 0 but not counter 1.
- Other uncore C-Box events can be programmed with either counter 0 or 1.

The C-Box uncore performance events can collect performance characteristics of transactions initiated by processor core. In that respect, they are similar to various sub-events in the OFFCORE_RESPONSE family of performance events in the core PMU. Information such as data supplier locality (LLC HIT/MISS) and snoop responses can be collected via OFFCORE_RESPONSE and qualified on a per-thread basis.

On the other hand, uncore performance event logic cannot associate its counts with the same level of per-thread qualification attributes as the core PMU events can. Therefore, whenever similar event programming capabilities are available from both core PMU and uncore PMU, the recommendation is that utilizing the core PMU events may be less affected by artifacts, complex interactions and other factors.

22.3.4.7 Intel® Xeon® Processor E5 Family Performance Monitoring Facility

The Intel® Xeon® Processor E5 Family (and Intel® Core™ i7-3930K Processor) are based on Sandy Bridge-E microarchitecture. While the processor cores share the same microarchitecture as those of the Intel® Xeon® Processor E3 Family and 2nd generation Intel Core i7-2xxx, Intel Core i5-2xxx, Intel Core i3-2xxx processor series, the uncore subsystems are different. An overview of the uncore performance monitoring facilities of the Intel Xeon processor E5 family (and Intel Core i7-3930K processor) is described in Section 22.3.4.8.

Thus, the performance monitoring facilities in the processor core generally are the same as those described in Section 22.6.3 through Section 22.3.4.5. However, the MSR_OFFCORE_RSP_0/MSR_OFFCORE_RSP_1 Response Supplier Info field shown in Table 22-18 applies to Intel Core Processors with CPUID signature of DisplayFamily_DisplayModel encoding of 06_2AH; Intel Xeon processor with CPUID signature of DisplayFamily_DisplayModel encoding of 06_2DH supports an additional field for remote DRAM controller shown in Table 22-21. Additionally, there are some small differences in the non-architectural performance monitoring events (see event list available at: <https://perfmon-events.intel.com/>).

Table 22-21. MSR_OFFCORE_RSP_x Supplier Info Field Definitions

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	LLC_HITM	18	M-state initial lookup stat in L3.
	LLC_HITE	19	E-state
	LLC_HITS	20	S-state
	LLC_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Remote	30:23	Remote DRAM Controller (either all 0s or all 1s).

22.3.4.8 Intel® Xeon® Processor E5 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5-2600 product family has some similarities with those of the Intel Xeon processor E7 family. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope. For example, each Cbox caching agent has a set of local performance counters, and the power controller unit (PCU) has its own local performance counters. Up to 8 C-Box units are supported in the uncore sub-system.

Table Table 22-22. "Uncore PMU MSR Summary for Intel® Xeon® Processor E5 Family" summarizes the uncore PMU facilities providing MSR interfaces.

Table 22-22. Uncore PMU MSR Summary for Intel® Xeon® Processor E5 Family

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	4	44	Yes	per-box	None
PCU	1	4	48	Yes	per-box	Match/Mask
U-Box	1	2	44	Yes	uncore	None

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 family is available in "Intel® Xeon® Processor E5 Uncore Performance Monitoring Programming Reference Manual". The MSR-based uncore PMU interfaces are listed in Table 2-24.

22.3.5 3rd Generation Intel® Core™ Processor Performance Monitoring Facility

The 3rd generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200v2 product family are based on the Ivy Bridge microarchitecture. The performance monitoring facilities in the processor core generally are the same as those described in Section 22.6.3 through Section 22.3.4.5. The non-architectural performance monitoring events supported by the processor core can be found at: <https://perfmon-events.intel.com/>.

22.3.5.1 Intel® Xeon® Processor E5 v2 and E7 v2 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5 v2 and Intel Xeon Processor E7 v2 product families are based on the Ivy Bridge-E microarchitecture. There are some similarities with those of the Intel Xeon processor E5 family based on the Sandy Bridge microarchitecture. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope.

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 v2 and Intel Xeon Processor E7 v2 families are available in the "Intel® Xeon® Processor E5 v2 and E7 v2 Uncore Performance Monitoring Programming Reference Manual". The MSR-based uncore PMU interfaces are listed in Table 2-28.

22.3.6 4th Generation Intel® Core™ Processor Performance Monitoring Facility

The 4th generation Intel® Core™ processor and Intel® Xeon® processor E3-1200 v3 product family are based on the Haswell microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 22.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 22.2.3.

The core PMU's capability is similar to those described in Section 22.6.3 through Section 22.3.4.5, with some differences and enhancements summarized in Table Table 22-23. "Core PMU Comparison". Additionally, the core PMU provides some enhancement to support performance monitoring when the target workload contains instruction streams using Intel® Transactional Synchronization Extensions (TSX), see Section 22.3.6.5. For details of Intel TSX, see Chapter 16, "Programming with Intel® AVX10," of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

Table 22-23. Core PMU Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 22.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4 or (8 if a core not shared by two threads)	Use CPUID to determine # of counters. See Section 22.2.1.
PMI Overhead Mitigation	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling. - Freeze_while_SMM.	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling. - Freeze_while_SMM.	See Section 20.4.7.
Processor Event Based Sampling (PEBS) Events	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture" and Section 22.3.6.5.1.	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 22.3.4.4.2.	See Section 22.3.4.4.2.	
PEBS-Precise Store	No, replaced by Data Address profiling.	Section 22.3.4.4.3	
PEBS-PDIR	Yes (using precise INST_RETIRED.ALL)	Yes (using precise INST_RETIRED.ALL)	
PEBS-EventingIP	Yes	No	
Data Address Profiling	Yes	No	
LBR Profiling	Yes	Yes	
Call Stack Profiling	Yes, see Section 20.11.	No	Use LBR facility.
Off-core Response Event	MSR 1A6H and 1A7H; extended request and response types.	MSR 1A6H and 1A7H; extended request and response types.	
Intel TSX support for PerfMon	See Section 22.3.6.5.	No	

22.3.6.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 4th Generation Intel Core processor is similar to those in processors based on Sandy Bridge microarchitecture, with several enhanced features. The key components and differences of PEBS facility relative to Sandy Bridge microarchitecture is summarized in Table Table 22-24. "PEBS Facility Comparison".

Table 22-24. PEBS Facility Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7
PEBS Buffer Programming	Section 22.3.1.1.1	Section 22.3.1.1.1	Unchanged
IA32_PEBE_ENABLE Layout	Figure 22-17	Figure 22-31	

Table 22-24. PEBS Facility Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
PEBS record layout	Table Table 22-25. "PEBS Record Format for 4th Generation Intel Core Processor Family"; enhanced fields at offsets 98H, A0H, A8H, B0H.	Table Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family"; enhanced fields at offsets 98H, A0H, A8H.	
Precise Events	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table Table 22-14. "Layout of Data Source Field of Load Latency Record".	Table Table 22-14. "Layout of Data Source Field of Load Latency Record"	
PEBS-Precise Store	No, replaced by data address profiling.	Yes; see Section 22.3.4.4.3.	
PEBS-PDIR	Yes	Yes	IA32_PMC1 only.
PEBS skid from EventingIP	1 (or 2 if micro+macro fusion)	1	
SAMPLING Restriction	Small SAV(CountDown) value incur higher overhead than prior generation.		

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTE

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

22.3.6.2 PEBS Data Format

The PEBS record format for the 4th Generation Intel Core processor is shown in Table Table 22-25. "PEBS Record Format for 4th Generation Intel Core Processor Family". The PEBS record format, along with debug/store area storage format, does not change regardless of whether IA-32e mode is active or not. CPUID.01H:ECX.DTES64[2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 22-25. PEBS Record Format for 4th Generation Intel Core Processor Family

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS

Table 22-25. PEBS Record Format for 4th Generation Intel Core Processor Family

Byte Offset	Field	Byte Offset	Field
38H	R/EDI	98H	Data Linear Address
40H	R/EBP	A0H	Data Source Encoding
48H	R/ESP	A8H	Latency value (core cycles)
50H	R8	B0H	EventingIP
58H	R9	B8H	TX Abort Information (Section 22.3.6.5.1)

The layout of PEBS records are almost identical to those shown in Table 22-4. "PEBS Record Format for Intel Core i7 Processor Family". Offset B0H is a new field that records the eventing IP address of the retired instruction that triggered the PEBS assist.

The PEBS records at offsets 98H, A0H, and ABH record data gathered from three of the PEBS capabilities in prior processor generations: load latency facility (Section 22.3.4.4.2), PDIR (Section 22.3.4.4.4), and the equivalent capability of precise store in prior generation (see Section 22.3.6.3).

In the core PMU of the 4th generation Intel Core processor, load latency facility and PDIR capabilities are unchanged. However, precise store is replaced by an enhanced capability, data address profiling, that is not restricted to store address. Data address profiling also records information in PEBS records at offsets 98H, A0H, and ABH.

22.3.6.3 PEBS Data Address Profiling

The Data Linear Address facility is also abbreviated as DataLA. The facility is a replacement or extension of the precise store facility in previous processor generations. The DataLA facility complements the load latency facility by providing a means to profile load and store memory references in the system, leverages the PEBS facility, and provides additional information about sampled loads and stores. Having precise memory reference events with linear address information for both loads and stores provides information to improve data structure layout, eliminate remote node references, and identify cache-line conflicts in NUMA systems.

The DataLA facility in the 4th generation processor supports the following events configured to use PEBS:

Table 22-26. Precise Events That Supports Data Linear Address Profiling

Event Name	Event Name
MEM_UOPS_RETIRE.STLB_MISS_LOADS	MEM_UOPS_RETIRE.STLB_MISS_STORES
MEM_UOPS_RETIRE.LOCK_LOADS	MEM_UOPS_RETIRE.SPLIT_STORES
MEM_UOPS_RETIRE.SPLIT_LOADS	MEM_UOPS_RETIRE.ALL_STORES
MEM_UOPS_RETIRE.ALL_LOADS	MEM_LOAD_UOPS_LLC_MISS_RETIRE.LOCAL_DRAM
MEM_LOAD_UOPS_RETIRE.L1_HIT	MEM_LOAD_UOPS_RETIRE.L2_HIT
MEM_LOAD_UOPS_RETIRE.L3_HIT	MEM_LOAD_UOPS_RETIRE.L1_MISS
MEM_LOAD_UOPS_RETIRE.L2_MISS	MEM_LOAD_UOPS_RETIRE.L3_MISS
MEM_LOAD_UOPS_RETIRE.HIT_LFB	MEM_LOAD_UOPS_L3_HIT_RETIRE.XSNP_MISS
MEM_LOAD_UOPS_L3_HIT_RETIRE.XSNP_HIT	MEM_LOAD_UOPS_L3_HIT_RETIRE.XSNP_HITM
UOPS_RETIRE.ALL (if load or store is tagged)	MEM_LOAD_UOPS_LLC_HIT_RETIRE.XSNP_NONE

DataLA can use any one of the IA32_PMC0-IA32_PMC3 counters. Counter overflows will initiate the generation of PEBS records. Upon counter overflow, hardware captures the linear address and possible other status information of the retiring memory uop. This information is then written to the PEBS record that is subsequently generated.

To enable the DataLA facility, software must complete the following steps. Please note that the DataLA facility relies on the PEBS facility, so the PEBS configuration requirements must be completed before attempting to capture DataLA information.

- Complete the PEBS configuration steps.
- Program an event listed in Table 22-26 using any one of IA32_PERFEVTSEL0-IA32_PERFEVTSEL3.
- Set the corresponding IA32_PEBS_ENABLE.PEBS_EN_CTRx bit. This enables the corresponding IA32_PMCx as a PEBS counter and enables the DataLA facility.

When the DataLA facility is enabled, the relevant information written into a PEBS record affects entries at offsets 98H, A0H, and A8H, as shown in Table 22-27. "Layout of Data Linear Address Information In PEBS Record".

Table 22-27. Layout of Data Linear Address Information In PEBS Record

Field	Offset	Description
Data Linear Address	98H	The linear address of the load or the destination of the store.
Store Status	A0H	▪ DCU Hit (Bit 0): The store hit the data cache closest to the core (L1 cache) if this bit is set, otherwise the store missed the data cache. This information is valid only for the following store events: UOPS_RETIRED.ALL (if store is tagged), MEM_UOPS_RETIRED.STLB_MISS_STORES, MEM_UOPS_RETIRED.SPLIT_STORES, MEM_UOPS_RETIRED.ALL_STORES ▪ Other bits are zero, The STLB_MISS, LOCK bit information can be obtained by programming the corresponding store event in Table 22-26. "Precise Events That Supports Data Linear Address Profiling".
Reserved	A8H	Always zero.

22.3.6.3.1 EventingIP Record

The PEBS record layout for processors based on Haswell microarchitecture adds a new field at offset 0B0H. This is the eventingIP field that records the IP address of the retired instruction that triggered the PEBS assist. The EIP/RIP field at offset 08H records the IP address of the next instruction to be executed following the PEBS assist.

22.3.6.4 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 22.3.4.5. The event codes are listed in Table 22-16. "Off-Core Response Event Encoding". Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 22-28. "MSR_OFFCORE_RSP_x Request_Type Definition (Haswell Microarchitecture)".
- Supplier information (bits 30:16): see Table 22-29. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signatures: 06_3CH, 06_46H)".
- Snoop response information (bits 37:31): see Table 22-19. "MSR_OFFCORE_RSP_x Snoop Info Field Definition".

Table 22-28. MSR_OFFCORE_RSP_x Request_Type Definition (Haswell Microarchitecture)

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts demand read (RFO) and software prefetches (PREFETCHW) for exclusive ownership in anticipation of a write.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
COREWB	3	Counts the number of modified cachelines written back.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.

Table 22-28. MSR_OFFCORE_RSP_x Request_Type Definition (Haswell Microarchitecture) (Contd.)

Bit Name	Offset	Description
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PF_L3_DATA_RD	7	Counts the number of data cacheline reads generated by L3 prefetchers.
PF_L3_RFO	8	Counts the number of RFO requests generated by L3 prefetchers.
PF_L3_CODE_RD	9	Counts the number of code reads generated by L3 prefetchers.
SPLIT_LOCK_UC_LOCK	10	Counts the number of lock requests that split across two cachelines or are to UC memory.
STRM_ST	11	Counts the number of streaming store requests electronically.
Reserved	14:12	Reserved
OTHER	15	Any other request that crosses IDI, including I/O.

The supplier information field listed in Table Table 22-29. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPIID Signatures: 06_3CH, 06_46H)". The fields vary across products (according to CPIID signatures) and is noted in the description.

Table 22-29. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPIID Signatures: 06_3CH, 06_46H)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	Reserved	21	Reserved
	LOCAL	22	Local DRAM Controller.
	Reserved	30:23	Reserved

Table 22-30. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPIID Signature: 06_45H)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	Reserved	21	Reserved
	L4_HIT_LOCAL_L4	22	L4 Cache
	L4_HIT_REMOTE_HOP0_L4	23	L4 Cache
	L4_HIT_REMOTE_HOP1_L4	24	L4 Cache
	L4_HIT_REMOTE_HOP2P_L4	25	L4 Cache
	Reserved	30:26	Reserved

22.3.6.4.1 Off-core Response Performance Monitoring in Intel Xeon Processors E5 v3 Series

Table 22-29. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signatures: 06_3CH, 06_46H)" lists the supplier information field that apply to Intel Xeon processor E5 v3 series (CPUID signature 06_3FH).

Table 22-31. MSR_OFFCORE_RSP_x Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	L3_HITF	21	F-state
	LOCAL	22	Local DRAM Controller.
	Reserved	26:23	Reserved
	L3_MISS_REMOTE_HOP0	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P	29	Hop 2 or more Remote supplier.
	Reserved	30	Reserved

22.3.6.5 Performance Monitoring and Intel® TSX

Chapter 16 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, describes the details of Intel® Transactional Synchronization Extensions (Intel® TSX). This section describes performance monitoring support for Intel TSX.

If a processor supports Intel TSX, the core PMU enhances its IA32_PERFEVTSELx MSR with two additional bit fields for event filtering. Support for Intel TSX is indicated by either (a) CPUID.07H.00H:EBX.RTM[11]=1, or (b) if CPUID.07H.00H:EBX.HLE[4] = 1. The TSX-enhanced layout of IA32_PERFEVTSELx is shown in Figure 22-36. The two additional bit fields are:

- **IN_TX** (bit 32): When set, the counter will only include counts that occurred inside a transactional region, regardless of whether that region was aborted or committed. This bit may only be set if the processor supports HLE or RTM.
- **IN_TXCP** (bit 33): When set, the counter will not include counts that occurred inside of an aborted transactional region. This bit may only be set if the processor supports HLE or RTM. This bit may only be set for IA32_PERFEVTSEL2.

When the IA32_PERFEVTSELx MSR is programmed with both IN_TX=0 and IN_TXCP=0 on a processor that supports Intel TSX, the result in a counter may include detectable conditions associated with a transaction code region for its aborted execution (if any) and completed execution.

In the initial implementation, software may need to take pre-caution when using the IN_TXCP bit. See Table 2-29.

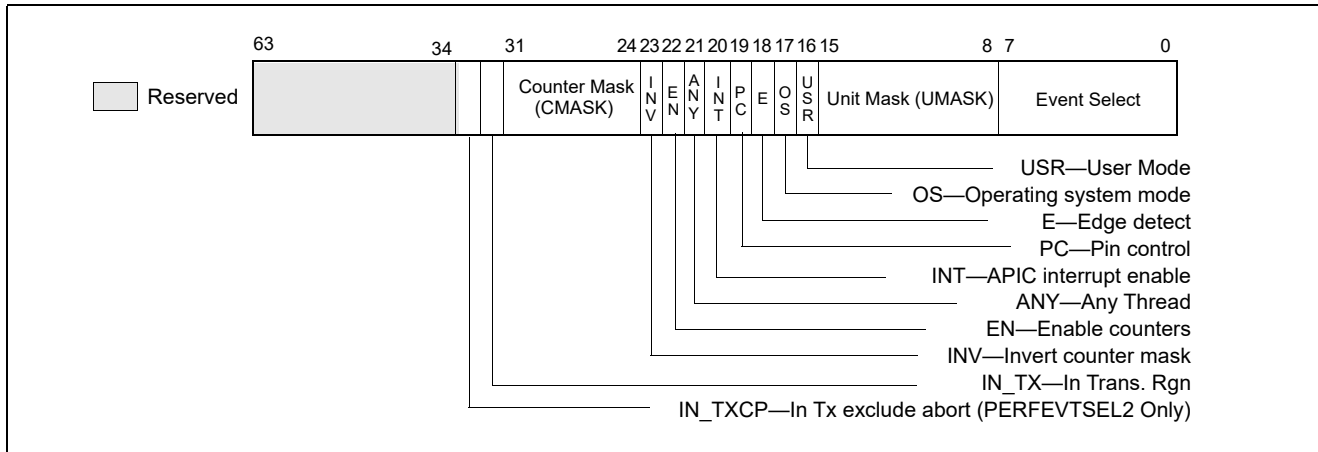


Figure 22-36. Layout of IA32_PERFEVTSELx MSRs Supporting Intel TSX

A common usage of setting `IN_TXCP=1` is to capture the number of events that were discarded due to a transactional abort. With `IA32_PMC2` configured to count in such a manner, then when a transactional region aborts, the value for that counter is restored to the value it had prior to the aborted transactional region. As a result, any updates performed to the counter during the aborted transactional region are discarded.

On the other hand, setting `IN_TX=1` can be used to drill down on the performance characteristics of transactional code regions. When a `PMCx` is configured with the corresponding `IA32_PERFEVTSELx.IN_TX=1`, only eventing conditions that occur inside transactional code regions are propagated to the event logic and reflected in the counter result. Eventing conditions specified by `IA32_PERFEVTSELx` but occurring outside a transactional region are discarded.

Additionally, a number of performance events are solely focused on characterizing the execution of Intel TSX transactional code, they can be found at: <https://perfmon-events.intel.com/>.

22.3.6.5.1 Intel® TSX and PEBS Support

If a PEBS event would have occurred inside a transactional region, then the transactional region first aborts, and then the PEBS event is processed.

Two of the TSX performance monitoring events also support using the PEBS facility to capture additional information. They are:

- `HLE_RETIRED.ABORTED` (encoding C8H mask 04H),
- `RTM_RETIRED.ABORTED` (encoding C9H mask 04H).

A transactional abort (`HLE_RETIRED.ABORTED`, `RTM_RETIRED.ABORTED`) can also be programmed to cause PEBS events. In this scenario, a PEBS event is processed following the abort.

Pending a PEBS record inside of a transactional region will cause a transactional abort. If a PEBS record was pended at the time of the abort or on an overflow of the TSX PEBS events listed above, only the following PEBS entries will be valid (enumerated by PEBS entry offset B8H bits[33:32] to indicate an HLE abort or an RTM abort):

- Offset B0H: EventingIP,
- Offset B8H: TX Abort Information

These fields are set for all PEBS events.

- Offset 08H (RIP/EIP) corresponds to the instruction following the outermost `XACQUIRE` in HLE or the first instruction of the fallback handler of the outermost `XBEGIN` instruction in RTM. This is useful to identify the aborted transactional region.

In the case of HLE, an aborted transaction will restart execution deterministically at the start of the HLE region. In the case of RTM, an aborted transaction will transfer execution to the RTM fallback handler.

The layout of the TX Abort Information field is given in Table Table 22-32. "TX Abort Information Field Definition".

Table 22-32. TX Abort Information Field Definition

Bit Name	Offset	Description
Cycles_Last_TX	31:0	The number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
HLE_Abort	32	If set, the abort information corresponds to an aborted HLE execution
RTM_Abort	33	If set, the abort information corresponds to an aborted RTM execution
Instruction_Abort	34	If set, the abort was associated with the instruction corresponding to the eventing IP (offset 0B0H) within the transactional region.
Non_Instruction_Abort	35	If set, the instruction corresponding to the eventing IP may not necessarily be related to the transactional abort.
Retry	36	If set, retrying the transactional execution may have succeeded.
Data_Conflict	37	If set, another logical processor conflicted with a memory address that was part of the transactional region that aborted.
Capacity Writes	38	If set, the transactional region aborted due to exceeding resources for transactional writes.
Capacity Reads	39	If set, the transactional region aborted due to exceeding resources for transactional reads.
In_Suspend	40	Transaction was aborted while in a suspend region. This is an Intel Xeon processor only feature, available beginning with 4th generation Intel Xeon Scalable Processor Family; otherwise reserved.
Reserved	63:41	Reserved

22.3.6.6 Uncore Performance Monitoring Facilities in the 4th Generation Intel® Core™ Processors

The uncore sub-system in the 4th Generation Intel® Core™ processors provides its own performance monitoring facility. The uncore PMU facility provides dedicated MSRs to select uncore performance monitoring events in a similar manner as those described in Section 22.3.4.6.

The ARB unit and each C-Box provide local pairs of event select MSR and counter register. The layout of the event select MSRs in the C-Boxes are identical as shown in Figure 22-34.

At the uncore domain level, there is a master set of control MSRs that centrally manages all the performance monitoring facility of uncore units. Figure 22-35 shows the layout of the uncore domain global control.

Additionally, there is also a fixed counter, counting uncore clockticks, for the uncore domain. Table 22-20. "Uncore PMU MSR Summary" summarizes the number MSRs for uncore PMU for each box.

Table 22-33. Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
C-Box	SKU specific	2	44	Yes	Per-box	Up to 4, see Table 2-21 MSR_UNC_CBO_CONFIG
ARB	1	2	44	Yes	Uncore	
Fixed Counter	N.A.	N.A.	48	No	Uncore	

The uncore performance events for the C-Box and ARB units can be found at: <https://perfmon-events.intel.com/>.

22.3.6.7 Intel® Xeon® Processor E5 v3 Family Uncore Performance Monitoring Facility

Details of the uncore performance monitoring facility of Intel Xeon Processor E5 v3 families are available in "Intel® Xeon® Processor E5 v3 Uncore Performance Monitoring Programming Reference Manual". The MSR-based uncore PMU interfaces are listed in Table 2-33.

22.3.7 5th Generation Intel® Core™ Processor and Intel® Core™ M Processor Performance Monitoring Facility

The 5th Generation Intel® Core™ processor and the Intel® Core™ M processor families are based on the Broadwell microarchitecture. The core PMU supports architectural performance monitoring capability with version ID 3 (see Section 22.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 22.2.3.

The core PMU has the same capability as those described in Section 22.3.6. IA32_PERF_GLOBAL_STATUS provide a bit indicator (bit 55) for PMI handler to distinguish PMI due to output buffer overflow condition due to accumulating packet data from Intel Processor Trace.

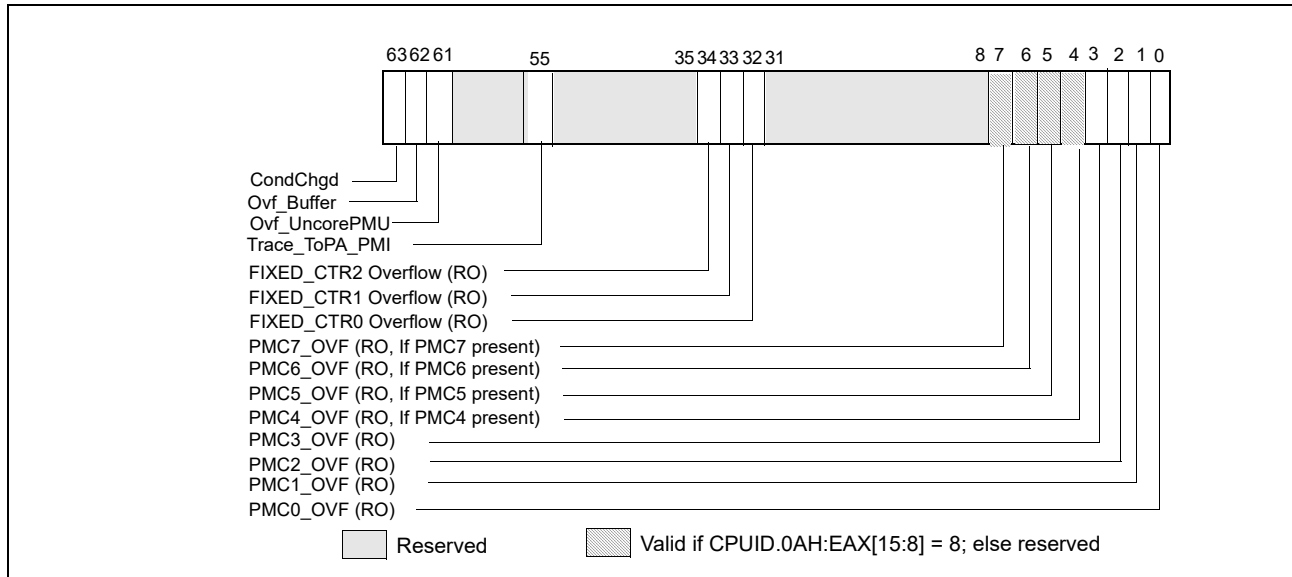


Figure 22-37. IA32_PERF_GLOBAL_STATUS MSR in Broadwell Microarchitecture

Details of Intel Processor Trace is described in Chapter 36, “Intel® Processor Trace.” The IA32_PERF_GLOBAL_OVF_CTRL MSR provides a corresponding reset control bit.

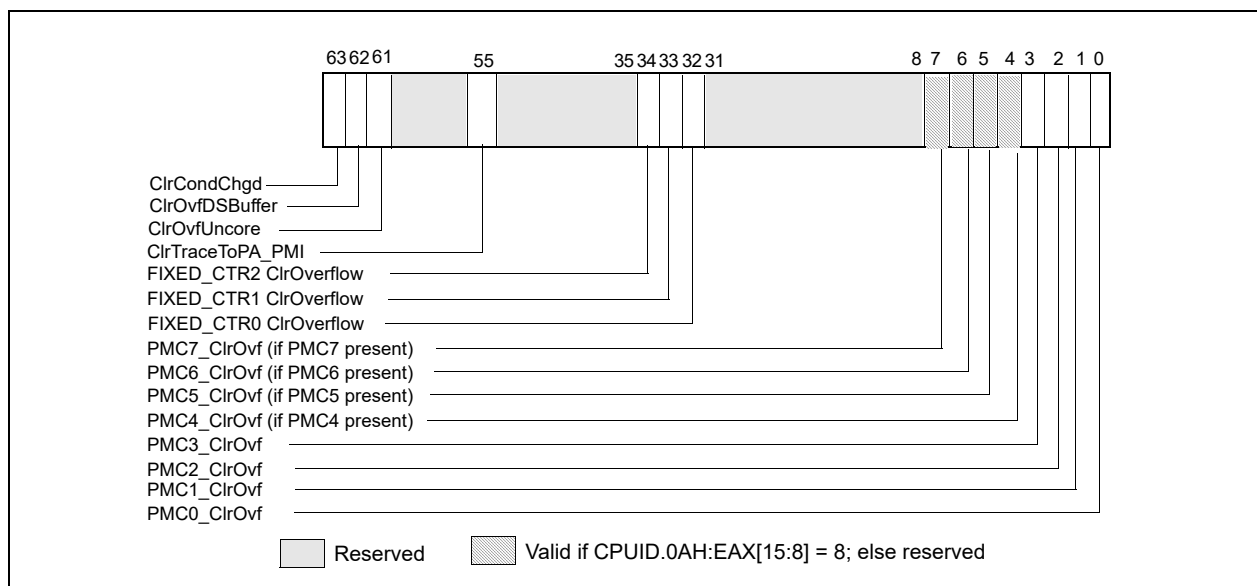


Figure 22-38. IA32_PERF_GLOBAL_OVF_CTRL MSR in Broadwell microarchitecture

The specifics of non-architectural performance events can be found at: <https://perfmon-events.intel.com/>.

22.3.8 6th Generation, 7th Generation and 8th Generation Intel® Core™ Processor Performance Monitoring Facility

The 6th generation Intel® Core™ processor is based on the Skylake microarchitecture. The 7th generation Intel® Core™ processor is based on the Kaby Lake microarchitecture. The 8th generation Intel® Core™ processors, 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on the Coffee Lake microarchitecture. For these microarchitectures, the core PMU supports architectural performance monitoring capability with version ID 4 (see Section 22.2.4) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 4 capabilities are described in Section 22.2.4.

The core PMU's capability is similar to those described in Section 22.6.3 through Section 22.3.4.5, with some differences and enhancements summarized in Table 22-34. "Core PMU Comparison". Additionally, the core PMU provides some enhancement to support performance monitoring when the target workload contains instruction streams using Intel® Transactional Synchronization Extensions (TSX), see Section 22.3.6.5. For details of Intel TSX, see Chapter 16, "Programming with Intel® AVX10," of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

Performance monitoring result may be affected by side-band activity on processors that support Intel SGX, details are described in Chapter 43, "Enclave Code Debug and Profiling."

Table 22-34. Core PMU Comparison

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
# of Fixed counters per thread	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 22.2.2.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4 or (8 if a core not shared by two threads)	Use CPUID to determine # of counters. See Section 22.2.1.
Architectural PerfMon version	4	3	See Section 22.2.4
PMI Overhead Mitigation	- Freeze_PerfMon_on_PMI with streamlined semantics. - Freeze_LBR_on_PMI with streamlined semantics. - Freeze_while_SMM.	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling. - Freeze_while_SMM.	See Section 20.4.7. Legacy semantics not supported with version 4 or higher.
Counter and Buffer Overflow Status Management	- Query via IA32_PERF_GLOBAL_STATUS - Reset via IA32_PERF_GLOBAL_STATUS_RESET - Set via IA32_PERF_GLOBAL_STATUS_SET	- Query via IA32_PERF_GLOBAL_STATUS - Reset via IA32_PERF_GLOBAL_OVF_CTRL	See Section 22.2.4.

Table 22-34. Core PMU Comparison (Contd.)

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
IA32_PERF_GLOBAL_STATUS Indicators of Overflow/Overhead/Interference	- Individual counter overflow - PEBS buffer overflow - ToPA buffer overflow - CTR_Frz, LBR_Frz, ASCI	- Individual counter overflow - PEBS buffer overflow - ToPA buffer overflow (applicable to Broadwell microarchitecture)	See Section 22.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	- CTR_Frz - LBR_Frz	NA	See Section 22.2.4.1.
PerfMon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	NA	See Section 22.2.4.3.
Precise Events	See Table Table 22-37. "Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures".	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	IA32_PMC4-PMC7 do not support PEBS.
PEBS for front end events	See Section 22.3.8.2.	No	
LBR Record Format Encoding	000101b	000100b	Section 20.4.8.1
LBR Size	32 entries	16 entries	
LBR Entry	From_IP/To_IP/LBR_Info triplet	From_IP/To_IP pair	Section 20.12
LBR Timing	Yes	No	Section 20.12.1
Call Stack Profiling	Yes, see Section 20.11	Yes, see Section 20.11	Use LBR facility.
Off-core Response Event	MSR 1A6H and 1A7H; Extended request and response types.	MSR 1A6H and 1A7H; Extended request and response types.	
Intel TSX support for PerfMon	See Section 22.3.6.5.	See Section 22.3.6.5.	

22.3.8.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 6th generation, 7th generation and 8th generation Intel Core processors provides a number enhancement relative to PEBS in processors based on Haswell/Broadwell microarchitectures. The key components and differences of PEBS facility relative to Haswell/Broadwell microarchitecture is summarized in Table Table 22-35. "PEBS Facility Comparison".

Table 22-35. PEBS Facility Comparison

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
Valid IA32_PMCx	PMC0-PMC3	PMC0-PMC3	No PEBS on PMC4-PMC7.
PEBS Buffer Programming	Section 22.3.1.1.1	Section 22.3.1.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 22-17	Figure 22-17	
PEBS-EventingIP	Yes	Yes	
PEBS record format encoding	0011b	0010b	
PEBS record layout	Table Table 22-36. "PEBS Record Format for the 6th Generation, 7th Generation, and 8th Generation Intel Core Processor Families"; enhanced fields at offsets 98H- B8H; and TSC record field at C0H.	Table Table 22-25. "PEBS Record Format for 4th Generation Intel Core Processor Family"; enhanced fields at offsets 98H, A0H, A8H, B0H.	

Table 22-35. PEBS Facility Comparison (Contd.)

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
Multi-counter PEBS resolution	PEBS record 90H resolves the eventing counter overflow.	PEBS record 90H reflects IA32_PERF_GLOBAL_STATUS.	
Precise Events	See Table Table 22-37. "Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures".	See Table Table 22-13. "PEBS Performance Events for Sandy Bridge Microarchitecture".	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-PDIR	Yes	Yes	IA32_PMC1 only.
PEBS-Load Latency	See Section 22.3.4.4.2.	See Section 22.3.4.4.2.	
Data Address Profiling	Yes	Yes	
FrontEnd event support	FrontEnd_Retried event and MSR_PEBS_FRONTEND.	No	IA32_PMC0-PMC3 only.

Only IA32_PMC0 through IA32_PMC3 support PEBS.

NOTES

PEBS events are only valid when the following fields of IA32_PERFEVTSELx are all zero: AnyThread, Edge, Invert, CMask.

In a PMU with PDIR capability, PEBS behavior is unpredictable if IA32_PERFEVTSELx or IA32_PMCx is changed for a PEBS-enabled counter while an event is being counted. To avoid this, changes to the programming or value of a PEBS-enabled counter should be performed when the counter is disabled.

22.3.8.1.1 PEBS Data Format

The PEBS record format for the 6th generation, 7th generation and 8th generation Intel Core processors is reporting with encoding 0011b in IA32_PERF_CAPABILITIES[11:8]. The lay out is shown in Table Table 22-36. "PEBS Record Format for the 6th Generation, 7th Generation, and 8th Generation Intel Core Processor Families". The PEBS record format, along with debug/store area storage format, does not change regardless of whether IA-32e mode is active or not. CPUID.01H:ECX.DTES64[2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 22-36. PEBS Record Format for the 6th Generation, 7th Generation, and 8th Generation Intel Core Processor Families

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	68H	R11
08H	R/EIP	70H	R12
10H	R/EAX	78H	R13
18H	R/EBX	80H	R14
20H	R/ECX	88H	R15
28H	R/EDX	90H	Applicable Counter
30H	R/ESI	98H	Data Linear Address
38H	R/EDI	A0H	Data Source Encoding
40H	R/EBP	A8H	Latency value (core cycles)

Table 22-36. PEBS Record Format for the 6th Generation, 7th Generation, and 8th Generation Intel Core Processor Families

Byte Offset	Field	Byte Offset	Field
48H	R/ESP	B0H	EventingIP
50H	R8	B8H	TX Abort Information (Section 22.3.6.5.1)
58H	R9	C0H	TSC
60H	R10		

The layout of PEBS records are largely identical to those shown in Table Table 22-25. "PEBS Record Format for 4th Generation Intel Core Processor Family".

The PEBS records at offsets 98H, A0H, and ABH record data gathered from three of the PEBS capabilities in prior processor generations: load latency facility (Section 22.3.4.4.2), PDIR (Section 22.3.4.4.4), and data address profiling (Section 22.3.6.3).

In the core PMU of the 6th generation, 7th generation and 8th generation Intel Core processors, load latency facility and PDIR capabilities and data address profiling are unchanged relative to the 4th generation and 5th generation Intel Core processors. Similarly, precise store is replaced by data address profiling.

With format 0010b, a snapshot of the IA32_PERF_GLOBAL_STATUS may be useful to resolve the situations when more than one of IA32_PMICx have been configured to collect PEBS data and two consecutive overflows of the PEBS-enabled counters are sufficiently far apart in time. It is also possible for the image at 90H to indicate multiple PEBS-enabled counters have overflowed. In the latter scenario, software cannot to correlate the PEBS record entry to the multiple overflowed bits.

With PEBS record format encoding 0011b, offset 90H reports the "applicable counter" field, which is a multi-counter PEBS resolution index allowing software to correlate the PEBS record entry with the eventing PEBS overflow when multiple counters are configured to record PEBS records. Additionally, offset C0H captures a snapshot of the TSC that provides a time line annotation for each PEBS record entry.

22.3.8.1.2 PEBS Events

The list of precise events supported for PEBS in the Skylake, Kaby Lake and Coffee Lake microarchitectures is shown in Table Table 22-37. "Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures".

Table 22-37. Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures

Event Name	Event Select	Sub-event	UMask
INST_RETIRED	C0H	PREC_DIST ¹	01H
		ALL_CYCLES ²	01H
OTHER_ASSISTS	C1H	ANY	3FH
BR_INST_RETIRED	C4H	CONDITIONAL	01H
		NEAR_CALL	02H
		ALL_BRANCHES	04H
		NEAR_RETURN	08H
		NEAR_TAKEN	20H
		FAR_BRACHES	40H
BR_MISP_RETIRED	C5H	CONDITIONAL	01H
		ALL_BRANCHES	04H
		NEAR_TAKEN	20H
FRONTEND_RETIRED	C6H	<Programmable ³ >	01H
HLE_RETIRED	C8H	ABORTED	04H
RTM_RETIRED	C9H	ABORTED	04H

Table 22-37. Precise Events for the Skylake, Kaby Lake, and Coffee Lake Microarchitectures (Contd.)

Event Name	Event Select	Sub-event	UMask
MEM_INST_RETIRED ²	D0H	LOCK_LOADS	21H
		SPLIT_LOADS	41H
		SPLIT_STORES	42H
		ALL_LOADS	81H
		ALL_STORES	82H
MEM_LOAD_RETIRED ⁴	D1H	L1_HIT	01H
		L2_HIT	02H
		L3_HIT	04H
		L1_MISS	08H
		L2_MISS	10H
		L3_MISS	20H
		HIT_LFB	40H
MEM_LOAD_L3_HIT_RETIRED ²	D2H	XSNP_MISS	01H
		XSNP_HIT	02H
		XSNP_HITM	04H
		XSNP_NONE	08H

NOTES:

1. Only available on IA32_PMC1.
2. INST_RETIRED.ALL_CYCLES is configured with additional parameters of cmask = 10 and INV = 1
3. Subevents are specified using MSR_PEBBS_FRONTEND, see Section 22.3.8.3
4. Instruction with at least one load uop experiencing the condition specified in the UMask.

22.3.8.1.3 Data Address Profiling

The PEBS Data address profiling on the 6th generation, 7th generation and 8th generation Intel Core processors is largely unchanged from the prior generation. When the DataLA facility is enabled, the relevant information written into a PEBS record affects entries at offsets 98H, A0H, and A8H, as shown in Table Table 22-27. "Layout of Data Linear Address Information In PEBS Record".

Table 22-38. Layout of Data Linear Address Information In PEBS Record

Field	Offset	Description
Data Linear Address	98H	The linear address of the load or the destination of the store.
Store Status	A0H	▪ DCU Hit (Bit 0): The store hit the data cache closest to the core (L1 cache) if this bit is set, otherwise the store missed the data cache. This information is valid only for the following store events: UOPS_RETIRED.ALL (if store is tagged), MEM_INST_RETIRED.STLB_MISS_STORES, MEM_INST_RETIRED.ALL_STORES, MEM_INST_RETIRED.SPLIT_STORES. ▪ Other bits are zero.
Reserved	A8H	Always zero.

22.3.8.2 Frontend Retired Facility

The Skylake Core PMU has been extended to cover common microarchitectural conditions related to the front end pipeline in addition to providing a generic latency mechanism that can locate fetch bubbles without necessarily attributing them to a particular condition. The facility counts the events if the associated instruction reaches retire-

ment (architecturally committed). Additionally, the user may opt to enable the PEBS facility to obtain precise information on the context of the event, e.g., EventingIP.

The supported frontend microarchitectural conditions require the following interfaces:

- The IA32_PERFEVTSELx MSR must select the FRONTEND_RETIRED event, EventSelect = C6H and UMASK = 01H.
- This event employs a new MSR, MSR_PEBS_FRONTEND, to specify the supported frontend event details, see Table Table 22-39. "FrontEnd_Retired Sub-Event Encodings Supported by MSR_PEBS_FRONTEND.EVTSEL".
- If precise information is desired, program the PEBS_EN_PMCx field of IA32_PEBS_ENABLE MSR as required.

Note the AnyThread field of IA32_PERFEVTSELx is ignored by the processor for the "FRONTEND_RETIRED" event.

The sub-event encodings supported by MSR_PEBS_FRONTEND.EVTSEL is given in Table Table 22-39. "FrontEnd_Retired Sub-Event Encodings Supported by MSR_PEBS_FRONTEND.EVTSEL".

Table 22-39. FrontEnd_Retired Sub-Event Encodings Supported by MSR_PEBS_FRONTEND.EVTSEL

Sub-Event Name	EVTSEL	Description
ANY_DSB_MISS	1H	Retired Instructions which experienced any decode stream buffer (DSB) miss.
DSB_MISS	11H	Retired Instructions which experienced a DSB miss that caused a fetch starvation cycle.
L1I_MISS	12H	The fetch of retired Instructions which experienced Instruction L1 Cache true miss ¹ . Additional requests to the same cache line as an in-flight L1I cache miss will not be counted.
L2_MISS	13H	The fetch of retired Instructions which experienced L2 Cache true miss. Additional requests to the same cache line as an in-flight MLC cache miss will not be counted.
ITLB_MISS	14H	The fetch of retired Instructions which experienced ITLB true miss. Additional requests to the same cache line as an in-flight ITLB miss will not be counted.
STLB_MISS	15H	The fetch of retired Instructions which experienced STLB true miss. Additional requests to the same cache line as an in-flight STLB miss will not be counted.
IDQ_READ_BUBBLES	6H	An IDQ read bubble is defined as any one of the 4 allocation slots of IDQ that is not filled by the front-end on any cycle where there is no back end stall. Using the threshold and latency fields in MSR_PEBS_FRONTEND allows counting of IDQ read bubbles of various magnitude and duration. Latency controls the number of cycles and Threshold controls the number of allocation slots that contain bubbles. The event counts if and only if a sequence of at least FE_LATENCY consecutive cycles contain at least FE_TRESHOLD number of bubbles each.

NOTES:

1. A true miss is the first miss for a cacheline/page (excluding secondary misses that fall into same cacheline/page).

The layout of MSR_PEBS_FRONTEND is given in Table Table 22-40. "MSR_PEBS_FRONTEND Layout".

Table 22-40. MSR_PEBS_FRONTEND Layout

Bit Name	Offset	Description
EVTSEL	7:0	Encodes the sub-event within FrontEnd_Retired that can use PEBS facility, see Table Table 22-39. "FrontEnd_Retired Sub-Event Encodings Supported by MSR_PEBS_FRONTEND.EVTSEL".
IDQ_Bubble_Length	19:8	Specifies the threshold of continuously elapsed cycles for the specified width of bubbles when counting IDQ_READ_BUBBLES event.
IDQ_Bubble_Width	22:20	Specifies the threshold of simultaneous bubbles when counting IDQ_READ_BUBBLES event.
Reserved	63:23	Reserved

The FRONTEND_RETIRED event is designed to help software developers identify exact instructions that caused front-end issues. There are some instances in which the event will, by design, the under-counting scenarios include the following:

- The event counts only retired (non-speculative) front-end events, i.e., events from just true program execution path are counted.
- The event will count once per cacheline (at most). If a cacheline contains multiple instructions which caused front-end misses, the count will be only 1 for that line.
- If the multibyte sequence of an instruction spans across two cachelines and causes a miss it will be recorded once. If there were additional misses in the second cacheline, they will not be counted separately.
- If a multi-uop instruction exceeds the allocation width of one cycle, the bubbles associated with these uops will be counted once per that instruction.
- If 2 instructions are fused (macro-fusion), and either of them or both cause front-end misses, it will be counted once for the fused instruction.
- If a front-end (miss) event occurs outside instruction boundary (e.g., due to processor handling of architectural event), it may be reported for the next instruction to retire.

22.3.8.3 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 22.3.4.5. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table Table 22-41. "MSR_OFFCORE_RSP_x Request_Type Definition (Skylake, Kaby Lake, and Coffee Lake Microarchitectures)".
- Supplier information (bits 29:16): see Table Table 22-42. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9EH)".
- Snoop response information (bits 37:30): see Table Table 22-43. "MSR_OFFCORE_RSP_x Snoop Info Field Definition (CUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9E, 06_55H)".

**Table 22-41. MSR_OFFCORE_RSP_x Request_Type Definition
(Skylake, Kaby Lake, and Coffee Lake Microarchitectures)**

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count hw or sw prefetches.
DMND_RFO	1	Counts the number of demand reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
Reserved	14:3	Reserved
OTHER	15	Counts miscellaneous requests, such as I/O and uncacheable accesses.

Table Table 22-42. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9EH)" lists the supplier information field that applies to 6th generation, 7th generation and 8th generation Intel Core processors. (6th generation Intel Core processor CUID signatures: 06_4EH and 06_5EH; 7th generation and 8th generation Intel Core processor CUID signatures: 06_8EH and 06_9EH).

**Table 22-42. MSR_OFFCORE_RSP_x Supplier Info Field Definition
(CUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9EH)**

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.

Table 22-42. MSR_OFFCORE_RSP_x Supplier Info Field Definition
(CPUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9EH)

Subtype	Bit Name	Offset	Description
Supplier Info	NO_SUPP	17	No Supplier Information available.
	L3_HITM	18	M-state initial lookup stat in L3.
	L3_HITE	19	E-state
	L3_HITS	20	S-state
	Reserved	21	Reserved
	L4_HIT	22	L4 Cache (if L4 is present in the processor).
	Reserved	25:23	Reserved
	DRAM	26	Local Node
	Reserved	29:27	Reserved
	SPL_HIT	30	L4 cache super line hit (if L4 is present in the processor).

Table Table 22-43. "MSR_OFFCORE_RSP_x Snoop Info Field Definition (CPUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9E, 06_55H)" lists the snoop information field that apply to processors with CPUID signatures 06_4EH, 06_5EH, 06_8EH, 06_9E, and 06_55H.

Table 22-43. MSR_OFFCORE_RSP_x Snoop Info Field Definition
(CPUID Signatures: 06_4EH, 06_5EH, 06_8EH, 06_9E, 06_55H)

Subtype	Bit Name	Offset	Description
Snoop Info	SPL_HIT	30	L4 cache super line hit (if L4 is present in the processor).
	SNOOP_NONE	31	No details on snoop-related information.
	SNOOP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNOOP_MISS	33	A snoop was needed and it missed all snooped caches: -For LLC Hit, ReslHitl was returned by all cores. -For LLC Miss, Rspl was returned by all sockets and data was returned from DRAM.
	SNOOP_HIT_NO_FWD	34	A snoop was needed and it hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. This includes: -Snoop Hit w/ Invalidation (LLC Hit, RFO). -Snoop Hit, Left Shared (LLC Hit/Miss, IFetch/Data_RD). -Snoop Hit w/ Invalidation and No Forward (LLC Miss, RFO Hit S). In the LLC Miss case, data is returned from DRAM.
	SNOOP_HIT_WITH_FWD	35	A snoop was needed and data was forwarded from a remote socket. This includes: -Snoop Forward Clean, Left Shared (LLC Hit/Miss, IFetch/Data_RD/RFT).
	SNOOP_HITM	36	A snoop was needed and it HitM-ed in local or remote cache. HitM denotes a cache-line was in modified state before effect as a results of snoop. This includes: -Snoop HitM w/ WB (LLC miss, IFetch/Data_RD). -Snoop Forward Modified w/ Invalidation (LLC Hit/Miss, RFO). -Snoop MtoS (LLC Hit, IFetch/Data_RD).
	SNOOP_NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.

22.3.8.3.1 Off-core Response Performance Monitoring for the Intel® Xeon® Scalable Processor Family

The following tables list the requestor and supplier information fields that apply to the Intel® Xeon® Scalable Processor Family.

- Transaction request type encoding (bits 15:0): see Table 22-44.
- Supplier information (bits 29:16): see Table 22-45.
- Supplier information (bits 29:16) with support for Intel® Optane™ DC Persistent Memory support: see Table 22-46. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature: 06_55H, Steppings 0x5H - 0xFH)".
- Snoop response information has not been changed and is the same as in (bits 37:30): see Table 22-43.

Table 22-44. MSR_OFFCORE_RSP_x Request_Type Definition (Intel® Xeon® Scalable Processor Family)

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts the number of demand data reads and page table entry cacheline reads. Does not count hw or sw prefetches.
DEMAND_RFO	1	Counts the number of demand reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DEMAND_CODE_RD	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
Reserved	3	Reserved.
PF_L2_DATA_RD	4	Counts the number of prefetch data reads into L2.
PF_L2_RFO	5	Counts the number of RFO Requests generated by the MLC prefetches to L2.
Reserved	6	Reserved.
PF_L3_DATA_RD	7	Counts the number of MLC data read prefetches into L3.
PF_L3_RFO	8	Counts the number of RFO requests generated by MLC prefetches to L3.
Reserved	9	Reserved.
PF_L1D_AND_SW	10	Counts data cacheline reads generated by hardware L1 data cache prefetcher or software prefetch requests.
Reserved	14:11	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O and un-cacheable accesses.

Table 22-45. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature: 06_55H)" lists the supplier information field that applies to the Intel Xeon Scalable Processor Family (CPUID signature: 06_55H).

Table 22-45. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature: 06_55H)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	SUPPLIER_NONE	17	No Supplier Information available.
	L3_HIT_M	18	M-state initial lookup stat in L3.
	L3_HIT_E	19	E-state
	L3_HIT_S	20	S-state
	L3_HIT_F	21	F-state
	Reserved	25:22	Reserved
	L3_MISS_LOCAL_DRAM	26	L3 Miss: local home requests that missed the L3 cache and were serviced by local DRAM.
	L3_MISS_REMOTE_HOP0_DRAM	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1_DRAM	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P_DRAM	29	Hop 2 or more Remote supplier.
	Reserved	30	Reserved

Table 22-46 lists the supplier information field that applies to the Intel Xeon Scalable Processor Family (CPUID signature: 06_55H, Steppings 0x5H - 0xFH).

Table 22-46. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature: 06_55H, Steppings 0x5H - 0xFH)

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Supplier Info	SUPPLIER_NONE	17	No Supplier Information available.
	L3_HIT_M	18	M-state initial lookup stat in L3.
	L3_HIT_E	19	E-state
	L3_HIT_S	20	S-state
	L3_HIT_F	21	F-state
	LOCAL_PMM	22	Local home requests that were serviced by local PMM.
	REMOTE_HOP0_PMM	23	Hop 0 Remote supplier.
	REMOTE_HOP1_PMM	24	Hop 1 Remote supplier.
	REMOTE_HOP2P_PMM	25	Hop 2 or more Remote supplier.
	L3_MISS_LOCAL_DRAM	26	L3 Miss: Local home requests that missed the L3 cache and were serviced by local DRAM.
	L3_MISS_REMOTE_HOP0_DRAM	27	Hop 0 Remote supplier.
	L3_MISS_REMOTE_HOP1_DRAM	28	Hop 1 Remote supplier.
	L3_MISS_REMOTE_HOP2P_DRAM	29	Hop 2 or more Remote supplier.
	Reserved	30	Reserved

22.3.8.4 Uncore Performance Monitoring Facilities on Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Cannon Lake microarchitecture introduces LLC support of up to six processor cores. To support six processor cores and eight LLC slices, existing MSRs have been rearranged and new CBo MSRs have been added. Uncore perfor-

mance monitoring software drivers from prior generations of Intel Core processors will need to update the MSR addresses. The new MSRs and updated MSR addresses have been added to the Uncore PMU listing in Section 2.17.2, “MSRs Specific to 8th Generation Intel® Core™ i3 Processors,” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

22.3.9 10th Generation Intel® Core™ Processor Performance Monitoring Facility

Some 10th generation Intel® Core™ processors and some 3rd generation Intel® Xeon® Scalable Processor Family are based on Ice Lake microarchitecture. Some 11th generation Intel® Core™ processors are based on the Tiger Lake microarchitecture, and some are based on the Rocket Lake microarchitecture. For these processors, the core PMU supports architectural performance monitoring capability with version Id 5 (see Section 22.2.5) and a host of non-architectural monitoring capabilities.

The core PMU's capability is similar to those described in Section 22.3.1 through Section 22.3.8, with some differences and enhancements summarized in Table 22-47.

Table 22-47. Core PMU Summary of the Ice Lake Microarchitecture

Box	Ice Lake Microarchitecture	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Comment
Architectural PerfMon version	5	4	See Section 22.2.5.
Number of programmable counters per thread	8	4	Use CPUID to determine number of counters. See Section 22.2.1.
PEBS: Basic functionality	Yes	Yes	See Section 22.3.9.1.
PEBS record format encoding	0100b	0011b	See Section 22.6.2.4.2.
Extended PEBS	PEBS is extended to all Fixed and General Purpose counters and to all performance monitoring events.	No	See Section 22.9.1.
Adaptive PEBS	Yes	No	See Section 22.9.2.
Performance Metrics	Yes (4)	No	See Section 22.3.9.3.
PEBS-PDIR	IA32_FIXED0 only (Corresponding counter control MSRs must be enabled.)	IA32_PMC1 only.	

22.3.9.1 Processor Event Based Sampling (PEBS) Facility

The PEBS facility in the 10th generation Intel Core processors provides a number of enhancements relative to PEBS in processors based on the Skylake, Kaby Lake, and Coffee Lake microarchitectures. Enhancement of the PEBS facility with Extended PEBS and Adaptive PEBS features is described in detail in Section 22.9.

The 3rd generation Intel Xeon Scalable Family of processors based on the Ice Lake microarchitecture introduce EPT-friendly PEBS. This allows EPT violations and other VM Exits to be taken on PEBS accesses to the DS Area. See Section 22.9.5 for details.

22.3.9.2 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 22.3.4.5. Each event code for off-core response monitoring requires programming an associated configuration MSR, MSR_OFFCORE_RSP_x. Software must program MSR_OFFCORE_RSP_x according to:

- Transaction request type encoding (bits 15:0): see Table 18-[N1].
- Response type encoding (bits 16-37) of

- Supplier information: see Table [18-N2].
- Snoop response information: see Table [18-N3].
- All transactions are tracked at cacheline granularity except some in request type OTHER.

**Table 22-48. MSR_OFFCORE_RSP_x Request_Type Definition
(Processors Based on Ice Lake Microarchitecture)**

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data and page table entry reads.
DEMAND_RFO	1	Counts demand read (RFO) and software prefetches (PREFETCHW) for exclusive ownership in anticipation of a write.
DEMAND_CODE_RD	2	Counts demand instruction fetches and instruction prefetches targeting the L1 instruction cache.
Reserved	3	Reserved
HWPF_L2_DATA_RD	4	Counts hardware generated data read prefetches targeting the L2 cache.
HWPF_L2_RFO	5	Counts hardware generated prefetches for exclusive ownership (RFO) targeting the L2 cache.
Reserved	6	Reserved
HWPF_L3	9:7 and 13 ¹	Counts hardware generated prefetches of any type targeting the L3 cache.
HWPF_L1D_AND_SWPF	10	Counts hardware generated data read prefetches targeting the L1 data cache and the following software prefetches (PREFETCHNTA, PREFETCHT0/1/2).
STREAMING_WR	11	Counts streaming stores.
Reserved	12	Reserved
Reserved	14	Reserved
OTHER	15	Counts miscellaneous requests, such as I/O and un-cacheable accesses.

NOTES:

1. All bits need to be set to 1 to count this type.

Ice Lake microarchitecture has added a new category of Response subtype, called a Combined Response Info. To count a feature in this type, all the bits specified must be set to 1.

A valid response type must be a non-zero value of the following expression:

Any | ['OR' of Combined Response Info Bits | [('OR' of Supplier Info Bits) & ('OR' of Snoop Info Bits)]]

If "ANY" bit[16] is set, other response type bits [17-39] are ignored.

Table 22-49 lists the supplier information field that applies to processors based on Ice Lake microarchitecture.

**Table 22-49. MSR_OFFCORE_RSP_x Supplier Info Field Definition
(Processors Based on Ice Lake Microarchitecture)**

Subtype	Bit Name	Offset	Description
Common	Any	16	Catch all value for any response types.
Combined Response Info	DRAM	26, 31, 32 ¹	Requests that are satisfied by DRAM.
	NON_DRAM	26, 37 ¹	Requests that are satisfied by a NON_DRAM system component. This includes MMIO transactions.
	L3_MISS	22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37 ¹	Requests that were not supplied by the L3 Cache. The event includes some currently reserved bits in anticipation of future memory designs.

**Table 22-49. MSR_OFFCORE_RSP_x Supplier Info Field Definition (Contd.)
(Processors Based on Ice Lake Microarchitecture)**

Subtype	Bit Name	Offset	Description
Supplier Info	L3_HIT	18,19, 20 ¹	Requests that hit in L3 cache. Depending on the snoop response the L3 cache may have retrieved the cacheline from another core's cache.
Reserved		17, 21:25, 27:29	Reserved.

NOTES:

1. All bits need to be set to 1 to count this type.

Table 22-50 lists the snoop information field that applies to processors based on Ice Lake microarchitecture.

**Table 22-50. MSR_OFFCORE_RSP_x Snoop Info Field Definition
(Processors Based on Ice Lake Microarchitecture)**

Subtype	Bit Name	Offset	Description
Snoop Info	Reserved	30	Reserved.
	SNOOP_NOT_NEEDED	32	No snoop was needed to satisfy the request.
	SNOOP_MISS	33	A snoop was sent and none of the snooped caches contained the cacheline.
	SNOOP_HIT_NO_FWD	34	A snoop was sent and hit in at least one snooped cache. The unmodified cacheline was not forwarded back, because the L3 already has a valid copy.
	Reserved	35	Reserved.
	SNOOP_HITM	36	A snoop was sent and the cacheline was found modified in another core's caches. The modified cacheline was forwarded to the requesting core.

22.3.9.3 Performance Metrics

The Ice Lake core PMU provides built-in support for Top-down Microarchitecture Analysis (TMA) method level 1 metrics. These metrics are always available to cross-validate performance observations, freeing general purpose counters to count other events in high counter utilization scenarios. For more details about the method, refer to Top-Down Analysis Method chapter (Appendix B.1) of the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

A new MSR called MSR_PERF_METRICS reports the metrics directly. Software can check (and/or expose to its guests) the availability of the PERF_METRICS feature using IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE (bit 15). For additional details on this MSR, refer to Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

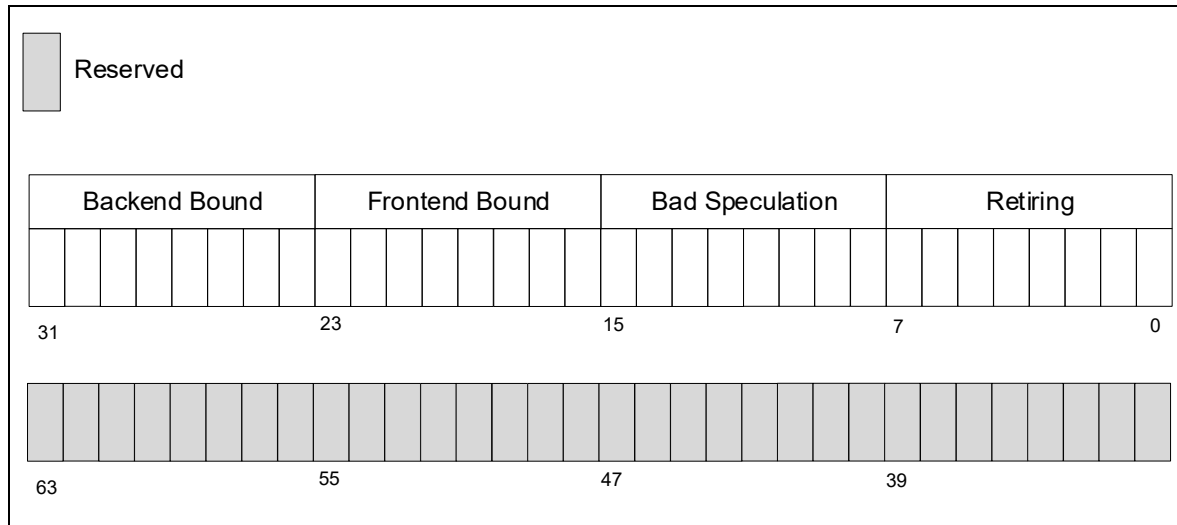


Figure 22-39. MSR_PERF_METRICS Definition

This register exposes the four TMA Level 1 metrics. The lower 32 bits are divided into four 8-bit fields, as shown by the above figure, each of which is an integer fraction of 255.

To support built-in performance metrics, new bits have been added to the following MSRs:

- IA32_PERF_GLOBAL_CTRL. EN_PERF_METRICS[48]: If this bit is set and fixed-function performance-monitoring counter 3 is enabled, built-in performance metrics are enabled.
- IA32_PERF_GLOBAL_STATUS_SET. SET_OVF_PERF_METRICS[48]: If this bit is set, it will set the status bit in the IA32_PERF_GLOBAL_STATUS register for PERF_METRICS.
- IA32_PERF_GLOBAL_STATUS_RESET. RESET_OVF_PERF_METRICS[48]: If this bit is set, it will clear the status bit in the IA32_PERF_GLOBAL_STATUS register for PERF_METRICS.
- IA32_PERF_GLOBAL_STATUS. OVF_PERF_METRICS[48]: If this bit is set, it indicates that a PERF_METRICS-related resource has overflowed and a PMI is triggered¹. If this bit is clear, no such overflow has occurred.

NOTE

Software has to synchronize, e.g., re-start, fixed-function performance-monitoring counter 3 as well as PERF_METRICS when either bit 35 or 48 in IA32_PERF_GLOBAL_STATUS is set. Otherwise, PERF_METRICS may return undefined values.

The values in MSR_PERF_METRICS are derived from fixed-function performance-monitoring counter 3. Software should start both registers, PERF_METRICS and fixed-function performance-monitoring counter 3, from zero. Additionally, software is recommended to periodically clear both registers in order to maintain accurate measurements for certain scenarios that involve sampling metrics at high rates.

In order to save/restore PERF_METRICS, software should follow these guidelines:

- PERF_METRICS and fixed-function performance-monitoring counter 3 should be saved and restored together.
- To ensure that PERF_METRICS and fixed-function performance-monitoring counter 3 remain synchronized, both should be disabled during both save and restore. Software should enable/disable them atomically, with a single write to IA32_PERF_GLOBAL_CTRL to set/clear both EN_PERF_METRICS[bit 48] and EN_FIXED_CTR3[bit 35].

1. An overflow of fixed-function performance-monitoring counter 3 should normally happen first if software follows Intel's recommendations.

- On state restore, fixed-function performance-monitoring counter 3 must be restored **before** PERF_METRICS, otherwise undefined results may be observed.

22.3.10 12th and 13th Generation Intel® Core™ Processors, and 4th and 5th Generation Intel® Xeon® Scalable Processor Family Performance Monitoring Facility

The 12th generation Intel® Core™ processor supports Alder Lake performance hybrid architecture. These processors offer a unique combination of Performance and Efficient-cores (P-core and E-core). The P-core is based on Golden Cove microarchitecture and the E-core is based on Gracemont microarchitecture. The 13th generation Intel® Core™ processor supports Raptor Lake performance hybrid architecture, utilizing both Raptor Cove cores and enhanced Gracemont cores. The 4th generation Intel® Xeon® Scalable Processor Family is based on Sapphire Rapids microarchitecture utilizing Golden Cove cores. The 5th generation Intel® Xeon® Scalable Processor Family is based on Sapphire Rapids microarchitecture utilizing Raptor Cove cores. These processors all report architectural performance monitoring version ID = 5 and support non-architectural monitoring capabilities described in this section.

22.3.10.1 P-core Performance Monitoring Unit

The P-core PMU's capability is similar to those described in Section 22.3.1 through Section 22.3.9, with some differences and enhancements summarized in Table 22-51.

Table 22-51. Core PMU Summary of the Golden Cove Microarchitecture

Box	Golden Cove Microarchitecture	Ice Lake Microarchitecture	Comment
Architectural PerfMon version	5	5	See Section 22.2.5.
Event-Counter Restrictions	Simplified identification		Counters 4-7 support a subset of events. See Section 22.3.10.1.2.
Performance Metrics	Yes (12)	Yes (4)	See Section 22.3.9.3.
PEBS: Baseline, record format	Yes 0100b	Yes 0100b	See Section 22.3.9.
PEBS: EPT-friendly	Yes	No; debuts in Ice Lake server microarchitecture	See Section 22.6.2.4.2.
PEBS: Precise Distribution	IA32_FIXED0 instruction-granularity PDist on IA32_PMC0	IA32_FIXED0 cycle-granularity No PDist	See Section 22.9.6.
PEBS: Load Latency	Instruction latency Cache latency Access info fields (5)	Instruction latency Access info fields (3)	See Section 22.9.7.
PEBS: Store Latency	Cache latency Access info fields (3)	None	See Section 22.9.8.
PEBS: Intel TSX support	Abort info fields (9)	Abort info fields (8)	See Section 22.3.6.5.1. (Intel Xeon processor only feature.)

22.3.10.1.1 P-core Perf Metrics Extensions

For 12th generation Intel Core processor P-cores, the core PMU supports the built-in metrics that were introduced in the Ice Lake microarchitecture PMU. This core PMU extends the PERF_METRICS MSR to feature TMA method level 2 metrics, as shown in Figure 22-40.

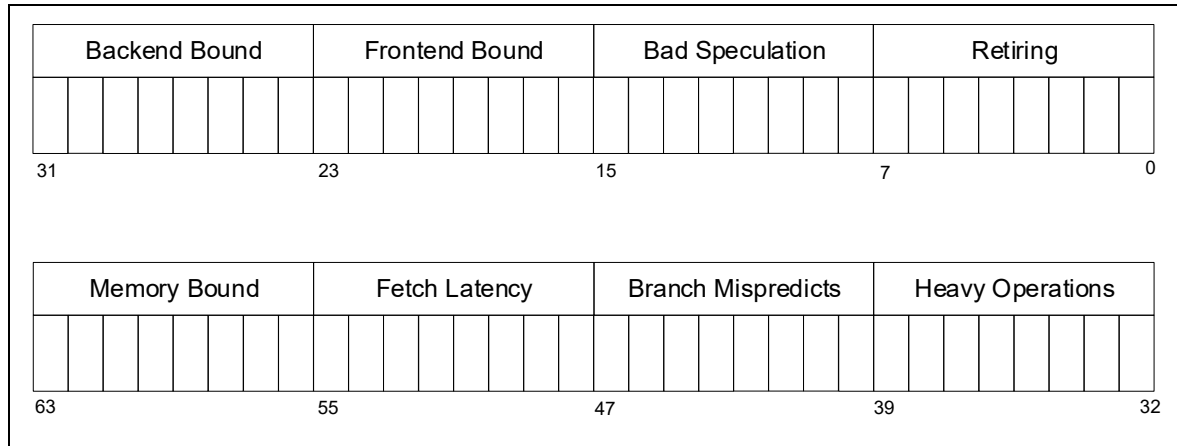


Figure 22-40. PERF_METRICS MSR Definition for 12th Generation Intel® Core™ Processor P-core

The lower half of the register is the TMA level 1 metrics (legacy). The upper half is also divided into four 8-bit fields, each of which is an integer fraction of 255. Additionally, each of the new level 2 metrics in the upper half is a subset of the corresponding level 1 metric in the lower half (that is, its parent node per the TMA hierarchy). This enables software to deduce the other four level 2 metrics by subtracting corresponding metrics as shown in Figure 22-41.

Light_Operations = Retiring - Heavy_Operations
 Machine_Clears = Bad_Speculation - Branch_Mispredicts
 Fetch_Bandwidth = Frontend_Bound - Fetch_Latency
 Core_Bound = Backend_Bound - Memory_Bound

Figure 22-41. Deducing Implied Level 2 Metrics in the Core PMU for 12th Generation Intel® Core™ Processor P-core

The PERF_METRICS MSR and fixed-function performance-monitoring counter 3 of the core PMU feature 12 metrics in total that cover all level 1 and level 2 nodes of the TMA hierarchy.

22.3.10.1.2 P-core Counter Restrictions Simplification

The 12th generation Intel Core processor P-core allows identification of performance monitoring events with counter restrictions based on event encodings. The general rule is: Event Codes < 0x90 are restricted to general-purpose performance-monitoring counters 0-3. Event Codes ≥ 0x90 are likely to have no restrictions. Table 22-52 lists the exceptions to this rule.

Table 22-52. Special Performance Monitoring Events with Counter Restrictions

Event Encoding ¹	Event Name	Counter Restriction
xx3C	CPU_CLK_UNHALTED.*	0-7 (No restriction for all architectural events.)
xx2E	LONGEST_LAT_CACHE.*	
xxDx	MEM_*_RETIRED.*	0-3
01A3, 02A3, 08A3	Some CYCLE_ACTIVITY sub-events	0-3
02CD	MEM_TRANS_RETIRED.STORE_SAMPLE	0

Table 22-52. Special Performance Monitoring Events with Counter Restrictions

Event Encoding ¹	Event Name	Counter Restriction
04A4	TOPDOWN.BAD_SPEC_SLOTS	0
08A4	TOPDOWN.BR_MISPREDICT_SLOTS	
xxCE	AMX_OPS_RETIRED	0

NOTES:

- Linux perf rUUEE syntax, where UU is the Unit Mask field and EE is the Event Select (also known as Event Code) field in the IA32_PERFECTSELx MSRs.

22.3.10.1.3 P-core Off-core Response Facility

For the 12th generation Intel Core processor P-core, the Off-core Response (OCR) Facility is similar to that described in Section 22.3.9.2.

The following enhancements are introduced for the Request_Type of MSR_OFFCORE_RSP_x:

- WB (bits 3 and 12): Count writeback (modified or non-modified) transactions by core caches.
- HWPf_L1D (bit 10): Counts hardware generated data read prefetches targeting the L1 data cache (only).
- SWPF_READ (bit 14): Counts software generated data read prefetches by the PREFETCHNTA and PREFETCHT0/1/2 instructions.

22.3.10.2 E-core Performance Monitoring Unit

The core PMU capabilities on the 12th generation Intel Core processor E-core are summarized in Table 22-53 below.

Table 22-53. Core PMU Summary of the Gracemont Microarchitecture

Box	Gracemont Microarchitecture	Tremont Microarchitecture	Comment
Number of fixed-function performance-monitoring counters per core	3	3	Use CPUID to enumerate number of counters. See Section 22.2.1.
Number of general-purpose counters per core	6	4	Use CPUID to enumerate number of counters. See Section 22.2.1.
Architectural Performance Monitoring version ID	5	5	See Section 22.2.5.
PEBS record format encoding	0100b	0100b	See Section 22.5.5.
EPT-friendly PEBS support	Yes	No	See Section 22.9.5.
Extended PEBS	Yes	Yes	See Section 22.9.1.
Adaptive PEBS	Yes	Yes	See Section 22.9.2.
Precise distribution (PDist) PEBS	IA32_PMC0 and IA32_FIXED_CTR0	IA32_PMC0 and IA32_FIXED_CTR0	PDist eliminates skid, see Section 22.9.3, Section 22.9.4, and Section 22.9.6.
PEBS Latency	Load and Store Latency	No	See Section 22.3.10.2.1, Section 22.3.10.2.2, Section 22.9.7, and Section 22.9.8.
PEBS Output	DS Save Area or Intel® Processor Trace	DS Save Area or Intel® Processor Trace	See Section 22.5.5.2.1.

Table 22-53. Core PMU Summary of the Gracemont Microarchitecture (Contd.)

Box	Gracemont Microarchitecture	Tremont Microarchitecture	Comment
Offcore Response	MSR 01A6H and 01A7H, each core has its own register, extended request and response types.	MSR 1A6H and 1A7H, each core has its own register, extended request and response types.	See Section 22.5.5.4.

22.3.10.2.1 E-core PEBS Load Latency

The 12th generation Intel Core processor E-core includes PEBS Load Latency support similar to that described in Section 22.9.7.

When a programmable counter is configured to count MEM_UOPS_RETIRED.LOAD_LATENCY_ABOVE_THRESHOLD (IA32_PERFVTSELx[15:0] = 0xD005, with CMASK=0 and INV=0), selected load operations whose latency exceeds the threshold provided in MSR_PEBS_LD_LAT_THRESHOLD (MSR 03F6H) will be counted. If a PEBS record is generated on overflow of this counter, the Memory Access Latency and Memory Auxiliary Info data is reported in the Memory Access Info group (Section 22.9.2.2.2). The formats of these fields are shown in Table 22-54 and Table 22-98.

Table 22-54. E-core PEBS Memory Access Info Encoding

Bit(s)	Field	Description
3:0	Data Source	The source of the data; see Table 22-55.
4	Lock	0: The operation was not part of a locked transaction. 1: The operation was part of a locked transaction.
5	STLB_MISS	0: The load did not miss the STLB (hit the DTLB or STLB). 1: The load missed the STLB.
6	ST_FWD_BLK	0: Load did not get a store forward block. 1: Load got a store forward block.
63:7	Reserved	Reserved

For details on E-core PEBS memory access latency encoding, see the Access Latency Field in Table 22-98.

Table 22-55. E-core PEBS Data Source Encodings

Encoding	Description
00H	Unknown Data Source (the processor could not retrieve the origin of this request) and MMIO. Memory mapped I/O hit.
01H	L1 HIT. This request was satisfied by the L1 data cache. (Minimal latency core cache hit.)
02H	FB HIT. Outstanding core cache miss to same cache-line address was already underway. (Pending core cache hit.)
03H	L2 HIT. This request was satisfied by the L2 cache.
04H	L3 HIT. Local or Remote home requests that hit L3 cache in the uncore with no coherency actions required (snooping).
05H	L3 HITE. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where no modified copies were found (clean).
06H	L3 HITM. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where a modified copy was found.
07H	Reserved.
08H	L3 HITF. Local or Remote home requests that hit the L3 cache and were serviced by another processor core with a cross core snoop where a shared or forwarding copy was found.
09H	Reserved.

Table 22-55. E-core PEBS Data Source Encodings (Contd.)

Encoding	Description
0AH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to shared state).
0BH	Reserved.
0CH	Reserved.
0DH	Reserved.
0EH	I/O. Request of input/output operation.
0FH	The request was to uncacheable memory.

22.3.10.2.2 E-core PEBS Store Latency

The 12th generation Intel Core processor E-core includes PEBS Store Latency support. When a programmable counter is configured to count MEM_UOPS_RETIRED.STORE_LATENCY (IA32_PERFVTSELx[15:0] = 0xD006, with CMASK=0 and INV=0), all store operations will be counted. If a PEBS record is generated on overflow of this counter, the Memory Access Latency and Memory Auxiliary Info data is reported in the Memory Access Info group (Section 22.9.2.2.2). The formats of these fields are shown in Table 22-54 and Table 22-98.

22.3.10.2.3 E-core Precise Distribution (PDist) Support

The 12th generation Intel Core processor E-core supports PEBS with Precise Distribution (PDist) on IA32_PMC0 and IA32_FIXED_CTR0. All precise events support PDist save for UOPS_RETIRED. See Section 22.9.6 for additional details on PDist.

22.3.10.2.4 E-core Enhanced Off-core Response

Event number 0B7H support off-core response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. There are unique pairs of MSR_OFFCORE_RSPx registers per core. The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as follows:

- Bits 15:0 and bits 49:44 specify the request type of a transaction request to the uncore. This is described in Table 22-56.
- Bits 30:16 specify Response Type information or an L2 Hit, and is described in Table 22-79.
- If L2 misses, then bits 37:31 can be used to specify snoop response information and is described in Table 22-80.
- For outstanding requests, bit 38 can enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously; see Section 22.5.2.3 for details.

Table 22-56. MSR_OFFCORE_RSPx Request Type Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data reads.
DEMAND_RFO	1	Counts all demand reads for ownership (RFO) requests and software based prefetches for exclusive ownership (prefetchw).
DEMAND_CODE_RD	2	Counts demand instruction fetches and L1 instruction cache prefetches.
COREWB_M	3	Counts modified write backs from L1 and L2.
HWPf_L2_DATA_RD	4	Counts prefetch (that bring data to L2) data reads.
HWPf_L2_RFO	5	Counts all prefetch (that bring data to L2) RFOs.
HWPf_L2_CODE_RD	6	Counts all prefetch (that bring data to MLC only) code reads.
HWPf_L3_DATA_RD	7	Counts L3 cache hardware prefetch data reads (written to the L3 cache only).

Table 22-56. MSR_OFFCORE_RSPx Request Type Definition

Bit Name	Offset	Description
HWPf_L3_RFO	8	Counts L3 cache hardware prefetch RFOs (written to the L3 cache only) .
HWPf_L3_CODE_RD	9	Counts L3 cache hardware prefetch code reads (written to the L3 cache only).
HWPf_L1D_AND_SWPF	10	Counts L1 data cache hardware prefetch requests, read for ownership prefetch requests and software prefetch requests (except prefetchw).
STREAMING_WR	11	Counts all streaming stores.
COREWB_NONM	12	Counts non-modified write backs from L2.
RSVD	14:13	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O accesses that have any response type.
UC_RD	44	Counts uncached memory reads (PRd, UCRdF).
UC_WR	45	Counts uncached memory writes (WiL).
PARTIAL_STREAMING_WR	46	Counts partial (less than 64 byte) streaming stores (WCiL).
FULL_STREAMING_WR	47	Counts full, 64 byte streaming stores (WCiLF).
L1WB_M	48	Counts modified WriteBacks from L1 that miss the L2.
L2WB_M	49	Counts modified WriteBacks from L2.

22.3.10.3 Unhalted Reference Cycles

The Unhalted Reference Cycles architectural performance monitoring event is enhanced to count at TSC-rate in the 12th generation Intel Core processor P-core when used on a general-purpose PMC. This enhancement makes it consistent with the fixed-function counter 2 and the E-core. As a result, this event is kept enumerated in CPUID.0AH.EBX (unlike prior hybrid parts).

22.3.11 Intel® Series 2 Core™ Ultra Processor Performance Monitoring Facility

The Intel® Series 2 Core™ Ultra processor supports Lunar Lake performance hybrid architecture. This processor offers a combination of Performance and Efficient-cores (P-core and E-core). The P-core is based on Lion Cove microarchitecture and the E-core is based on Skymont microarchitecture. This processor reports architectural performance monitoring version ID = 6 and supports non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 6 capabilities are described in Section 22.2.6.

22.3.11.1 P-core Performance Monitoring Unit

The core PMU capabilities on the Intel Series 2 Core Ultra processor P-core are similar to those described in Section 22.3.1 through Section 22.3.10, with some differences and enhancements summarized in Table 22-57.

Table 22-57. Core PMU Summary of the Lion Cove Microarchitecture

Box	Lion Cove Microarchitecture	Golden Cove and Redwood Cove Microarchitectures	Comment
Architectural Performance Monitoring version ID	6	5	See Section 22.2.6.
Number of general-purpose counters per core	10	8	Use CPUID to enumerate number of counters. See Section 22.2.1 and Section 22.2.9.

Table 22-57. Core PMU Summary of the Lion Cove Microarchitecture (Contd.)

Box	Lion Cove Microarchitecture	Golden Cove and Redwood Cove Microarchitectures	Comment
Number of architectural performance-monitoring events	12	8 (Golden Cove) 11 (Redwood Cove)	See Section 22.2.7.
Event-Counter Restrictions	Mostly homogeneous general counters.	Simplified identification of events supported on counters 4-7.	Few counter restrictions may apply; see Section 22.3.1.1.1.
Performance Metrics	12 metrics Metrics clear mode or read-only.	12 metrics Read-only.	See the RDPMC instruction in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B.
OCR: MSR_OFFCORE_RSP_0/1	0FFF FFFF FFFFH	003F FFFF FFFFH	See Section 2.17.9 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.
PEBS: Baseline	Yes	Yes	See Section 22.8.
PEBS record format encoding	0110b	0101b	See Section 22.9.2.2.
PEBS: Precise Distribution	IA32_FIXED0 instruction-granularity. PDist on IA32_PMC0 and IA32_PMC1.	IA32_FIXED0 instruction-granularity. PDist on IA32_PMC0.	See Section 22.9.6.
PEBS: Data Source field	5-bits	4-bits	See Section 22.9.7.
LBR: Event Logging	Yes	No	See Section 21.1.3.6.
Intel PT: TNT Disable	Yes	No	See Chapter 36 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

22.3.11.1.1 P-core Homogeneous General Counters

The Lion Cove PMU enhances general-counters to support most of the performance monitoring events. The remaining events that do have counter restrictions are summarized in next Table 22-58.

Table 22-58. Performance Monitoring Events with Counter Restrictions in Lion Cove PMU

Event Encoding ¹	Event Name	Counter Restriction
xx20	OFFCORE_REQUESTS_OUTSTANDING.*	0-3
0148	L1D_PENDING.*	2
0175	INST_DECODED.DECODERS	2
08A3, 0CA3	CYCLE_ACTIVITY.*_L1D_MISS	2
04A4, 08A4, 10A4	TOPDOWN.BAD_SPEC_SLOTS, TOPDOWN.BR_MISPREDICT_SLOTS, TOPDOWN.MEMORY_BOUND_SLOTS	0
01B1	UOPS_EXECUTED.*	3
xxDx	MEM_INST_RETIRED.*, MEM_LOAD*_RETIRED.*	0-3
02CD	MEM_TRANS_RETIRED.STORE_SAMPLE	0-1

NOTES:

1. Linux perf rUUEE syntax, where UU is the Unit Mask field and EE is the Event Select (also known as Event Code) field in the IA32_PERFECTSELx MSRs.

22.3.11.2 E-core Performance Monitoring Unit

Skymont microarchitecture performance monitoring capabilities are similar to Crestmont microarchitecture capabilities, with the following extensions:

- Support for fixed counters 4, 5, and 6 (see Section 22.2.9.4)
- Architecturally defined events: TMA L1 and LBR Inserts (see Section 22.2.9.6 and Section 21.1.3.6)
- PEBS Counter Snapshotting (see Section 22.9.10)
- Auto Counter Reload (see Section 22.11)

The core PMU capabilities on the Intel Series 2 Core Ultra processor E-core are summarized in Table 22-59.

Table 22-59. Core PMU Summary of the Skymont Microarchitecture

Box	Skymont Microarchitecture	Crestmont Microarchitecture	Comment
Architectural Performance Monitoring version ID	6	5	See Section 22.2.6.
Number of fixed-function performance-monitoring counters per core	6 (0, 1, 2, 4, 5, 6)	3 (0, 1, 2)	Use CPUID to enumerate number of counters. See Section 22.2.1 and Section 22.2.9.4.
Number of general-purpose counters per core	8	8	Use CPUID to enumerate number of counters. See Section 22.2.1.
PEBS record format encoding	0110b	0101b	See Section 22.3.10.
Auto Counter Reload (ACR)	Yes	No	See Section 22.11.

22.4 PERFORMANCE MONITORING (INTEL® XEON™ PHI PROCESSORS)

NOTE

This section also applies to the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series based on Knights Mill microarchitecture.

22.4.1 Intel® Xeon Phi™ Processor 7200/5200/3200 Performance Monitoring

The Intel® Xeon Phi™ processor 7200/5200/3200 series are based on the Knights Landing microarchitecture. The performance monitoring capabilities are distributed between its tiles (pair of processor cores) and untile (connecting many tiles in a physical processor package). Functional details of the tiles and untile of the Knights Landing microarchitecture can be found in Chapter 16 of Intel® 64 and IA-32 Architectures Optimization Reference Manual.

A complete description of the tile and untile PMU programming interfaces for Intel Xeon Phi processors based on the Knights Landing microarchitecture can be found in the Technical Document section at <http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html>.

A tile contains a pair of cores attached to a shared L2 cache and is similar to those found in Intel Atom® processors based on the Silvermont microarchitecture. The processor provides several new capabilities on top of the Silvermont performance monitoring facilities.

The processor supports architectural performance monitoring capability with version ID 3 (see Section 22.2.3) and a host of non-architectural performance monitoring capabilities. The processor provides two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2).

Non-architectural performance monitoring in the processor also uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 22-6 and described in Section and Section 22.2.3. The processor supports AnyThread counting in three architectural performance monitoring events.

22.4.1.1 Enhancements of Performance Monitoring in the Intel® Xeon Phi™ Processor Tile

The Intel® Xeon Phi™ processor tile includes the following enhancements to the Silvermont microarchitecture.

- AnyThread support. This facility is limited to following three architectural events: Instructions Retired, Unhalted Core Cycles, Unhalted Reference Cycles using IA32_FIXED_CTR0-2 and Unhalted Core Cycles, Unhalted Reference Cycles using IA32_PERFEVTSELx.
- PEBS-DLA (Processor Event-Based Sampling-Data Linear Address) fields. The processor provides memory address in addition to the Silvermont PEBS record support on select events. The PEBS recording format as reported by IA32_PERF_CAPABILITIES [11:8] is 2.
- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor tile to subsystems outside the tile (untile). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx. Two cores do not share the off-core response MSRs. Knights Landing expands off-core response capability to match the processor untile changes.
- Average request latency measurement. The off-core response counting facility can be combined to use two performance counters to count the occurrences and weighted cycles of transaction requests. This facility is updated to match the processor untile changes.

22.4.1.1.1 Processor Event-Based Sampling

The processor supports processor event based sampling (PEBS). PEBS is supported using IA32_PMC0 (see also Section 20.4.9, “BTS and DS Save Area”).

PEBS uses a debug store mechanism to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 22.6.2.4).

The list of PEBS events supported in the processor is shown in the following table.

Table 22-60. PEBS Performance Events for Knights Landing Microarchitecture

Event Name	Event Select	Sub-event	UMask	Data Linear Address Support
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H	No
		JCC	7EH	No
		TAKEN_JCC	FEH	No
		CALL	F9H	No
		REL_CALL	FDH	No
		IND_CALL	FBH	No
		NON_RETURN_IND	EBH	No
		FAR_BRANCH	BFH	No
		RETURN	F7H	No

Table 22-60. PEBS Performance Events for Knights Landing Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask	Data Linear Address Support
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H	No
		JCC	7EH	No
		TAKEN_JCC	FEH	No
		IND_CALL	FBH	No
		NON_RETURN_IND	EBH	No
		RETURN	F7H	No
MEM_UOPS_RETIRED	04H	L2_HIT_LOADS	02H	Yes
		L2_MISS_LOADS	04H	Yes
		DLTB_MISS_LOADS	08H	Yes
RECYCLEQ	03H	LD_BLOCK_ST_FORWARD	01H	Yes
		LD_SPLITS	08H	Yes

The PEBS record format 2 supported by processors based on the Knights Landing microarchitecture is shown in Table Table 22-61. "PEBS Record Format for Knights Landing Microarchitecture", and each field in the PEBS record is 64 bits long.

Table 22-61. PEBS Record Format for Knights Landing Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS
38H	R/EDI	98H	PSDLA
40H	R/EBP	A0H	Reserved
48H	R/ESP	A8H	Reserved
50H	R8	B0H	EventingRIP
58H	R9	B8H	Reserved

22.4.1.1.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table Table 22-62. "OffCore Response Event Encoding" lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

Table 22-62. OffCore Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMC0-1	B7H	01H	MSR_OFFCORE_RSP0 (address 1A6H)
PMC0-1	B7H	02H	MSR_OFFCORE_RSP1 (address 1A7H)

Some of the MSR_OFFCORE_RESP [0,1] register bits are not valid in this processor and their use is reserved. The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 registers are defined in Table Table 22-63. "Bit fields of the MSR_OFFCORE_RESP [0, 1] Registers". Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

Additionally, MSR_OFFCORE_RSP0 provides bit 38 to enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously, see Section 22.5.2.3 for details.

Table 22-63. Bit fields of the MSR_OFFCORE_RESP [0, 1] Registers

Main	Sub-field	Bit	Name	Description
Request Type		0	DEMAND_DATA_RD	Demand cacheable data and L1 prefetch data reads.
		1	DEMAND_RFO	Demand cacheable data writes.
		2	DEMAND_CODE_RD	Demand code reads and prefetch code reads.
		3	Reserved	Reserved.
		4	Reserved	Reserved.
		5	PF_L2_RFO	L2 data RFO prefetches (includes PREFETCHW instruction).
		6	PF_L2_CODE_RD	L2 code HW prefetches.
		7	PARTIAL_READS	Partial reads (UC or WC).
		8	PARTIAL_WRITES	Partial writes (UC or WT or WP). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
		9	UC_CODE_READS	UC code reads.
		10	BUS_LOCKS	Bus locks and split lock requests.
		11	FULL_STREAMING_STORES	Full streaming stores (WC). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
		12	SW_PREFETCH	Software prefetches.
		13	PF_L1_DATA_RD	L1 data HW prefetches.
		14	PARTIAL_STREAMING_STORES	Partial streaming stores (WC). Valid only for OFFCORE_RESP_1 event. Should only be used on PMC1. This bit is reserved for OFFCORE_RESP_0 event.
		15	ANY_REQUEST	Account for any requests.
Response Type	Any	16	ANY_RESPONSE	Account for any response.
	Data Supply from Untile	17	NO_SUPP	No Supplier Details.
		18	Reserved	Reserved.
		19	L2_HIT_OTHER_TILE_NEAR	Other tile L2 hit E Near.
		20	Reserved	Reserved.
		21	MCDRAM_NEAR	MCDRAM Local.
		22	MCDRAM_FAR_OR_L2_HIT_OTHER_TILE_FAR	MCDRAM Far or Other tile L2 hit far.
		23	DRAM_NEAR	DRAM Local.
		24	DRAM_FAR	DRAM Far.

Table 22-63. Bit fields of the MSR_OFFCORE_RESP [0, 1] Registers (Contd.)

Main	Sub-field	Bit	Name	Description
	Data Supply from within same tile	25	L2_HITM_THIS_TILE	M-state.
		26	L2_HITE_THIS_TILE	E-state.
		27	L2_HITS_THIS_TILE	S-state.
		28	L2_HITF_THIS_TILE	F-state.
		29	Reserved	Reserved.
		30	Reserved	Reserved.
	Snoop Info; Only Valid in case of Data Supply from Untile	31	SNOOP_NONE	None of the cores were snooped.
		32	NO_SNOOP_NEEDED	No snoop was needed to satisfy the request.
		33	Reserved	Reserved.
		34	Reserved	Reserved.
		35	HIT_OTHER_TILE_FWD	Snoop request hit in the other tile with data forwarded.
		36	HITM_OTHER_TILE	A snoop was needed and it HitM-ed in other core's L1 cache. HitM denotes a cache-line was in modified state before effect as a result of snoop.
		37	NON_DRAM	Target was non-DRAM system address. This includes MMIO transactions.
Outstanding requests	Weighted cycles	38	OUTSTANDING (Valid only for MSR_OFFCORE_RESP0. Should only be used on PMCO. This bit is reserved for MSR_OFFCORE_RESP1).	If set, counts total number of weighted cycles of any outstanding offcore requests with data response. Valid only for OFFCORE_RESP_0 event. Should only be used on PMCO. This bit is reserved for OFFCORE_RESP_1 event.

22.4.1.1.3 Average Offcore Request Latency Measurement

Measurement of average latency of offcore transaction requests can be enabled using MSR_OFFCORE_RSP0.[bit 38] with the choice of request type specified in MSR_OFFCORE_RSP0.[bit 15:0].

Refer to Section 22.5.2.3, "Average Offcore Request Latency Measurement," for typical usage. Note that MSR_OFFCORE_RESPx registers are not shared between cores in Knights Landing. This allows one core to measure average latency while other core is measuring different offcore response events.

22.5 PERFORMANCE MONITORING (INTEL ATOM® PROCESSORS)

22.5.1 Performance Monitoring (45 nm and 32 nm Intel Atom® Processors)

45 nm and 32 nm Intel Atom processors report architectural performance monitoring versionID = 3 (supporting the aggregate capabilities of versionID 1, 2, and 3; see Section 22.2.3) and a host of non-architectural monitoring capabilities. These 45 nm and 32 nm Intel Atom processors provide two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2).

NOTE

The number of counters available to software may vary from the number of physical counters present on the hardware, because an agent running at a higher privilege level (e.g., a VMM) may

not expose all counters. CPUID.0AH:EAX[15:8] reports the MSRs available to software; see Section 22.2.1.

Non-architectural performance monitoring in Intel Atom processor family uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

Architectural and non-architectural performance monitoring events in 45 nm and 32 nm Intel Atom processors support thread qualification using bit 21 (AnyThread) of IA32_PERFEVTSELx MSR, i.e., if IA32_PERFEVTSELx.AnyThread = 1, event counts include monitored conditions due to either logical processors in the same processor core.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 22-6 and described in Section and Section 22.2.3.

Valid event mask (Umask) bits can be found at: <https://perfmon-events.intel.com/>. The UMASK field may contain sub-fields that provide the same qualifying actions like those listed in Table Table 22-81. "Core Specificity Encoding within a Non-Architectural Umask", Table Table 22-82. "Agent Specificity Encoding within a Non-Architectural Umask", Table Table 22-83. "HW Prefetch Qualification Encoding within a Non-Architectural Umask", and Table Table 22-84. "MESI Qualification Definitions within a Non-Architectural Umask". One or more of these sub-fields may apply to specific events on an event-by-event basis. Precise Event Based Monitoring is supported using IA32_PMC0 (see also Section 20.4.9, "BTS and DS Save Area").

22.5.2 Performance Monitoring for Silvermont Microarchitecture

Intel processors based on the Silvermont microarchitecture report architectural performance monitoring versionID = 3 (see Section 22.2.3) and a host of non-architectural monitoring capabilities. Intel processors based on the Silvermont microarchitecture provide two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2). Intel Atom processors based on the Airmont microarchitecture support the same performance monitoring capabilities as those based on the Silvermont microarchitecture.

Non-architectural performance monitoring in the Silvermont microarchitecture uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events can be found at: <https://perfmon-events.intel.com/>.

The bit fields (except bit 21) within each IA32_PERFEVTSELx MSR are defined in Figure 22-6 and described in Section and Section 22.2.3. Architectural and non-architectural performance monitoring events in the Silvermont microarchitecture ignore the AnyThread qualification regardless of its setting in IA32_PERFEVTSELx MSR.

22.5.2.1 Enhancements of Performance Monitoring in the Processor Core

The notable enhancements in the monitoring of performance events in the processor core include:

- The width of counter reported by CPUID.0AH:EAX[23:16] is 40 bits.
- Off-core response counting facility. This facility in the processor core allows software to count certain transaction responses between the processor core to sub-systems outside the processor core (uncore). Counting off-core response requires additional event qualification configuration facility in conjunction with IA32_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx.
- Average request latency measurement. The off-core response counting facility can be combined to use two performance counters to count the occurrences and weighted cycles of transaction requests.

22.5.2.1.1 Processor Event Based Sampling (PEBS)

In the Silvermont microarchitecture, the PEBS facility can be used with precise events. PEBS is supported using IA32_PMC0 (see also Section 20.4.9).

PEBS uses a debug store mechanism to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 22.6.2.4).

The list of precise events supported in the Silvermont microarchitecture is shown in Table Table 22-64. "PEBS Performance Events for the Silvermont Microarchitecture".

Table 22-64. PEBS Performance Events for the Silvermont Microarchitecture

Event Name	Event Select	Sub-event	UMask
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		CALL	F9H
		REL_CALL	FDH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		FAR_BRANCH	BFH
		RETURN	F7H
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		RETURN	F7H
MEM_UOPS_RETIRED	04H	L2_HIT_LOADS	02H
		L2_MISS_LOADS	04H
		DLTB_MISS_LOADS	08H
		HITM	20H
REHABQ	03H	LD_BLOCK_ST_FORWARD	01H
		LD_SPLITS	08H

PEBS Record Format The PEBS record format supported by processors based on the Intel Silvermont microarchitecture is shown in Table Table 22-65. "PEBS Record Format for the Silvermont Microarchitecture", and each field in the PEBS record is 64 bits long.

Table 22-65. PEBS Record Format for the Silvermont Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	60H	R10
08H	R/EIP	68H	R11
10H	R/EAX	70H	R12
18H	R/EBX	78H	R13
20H	R/ECX	80H	R14
28H	R/EDX	88H	R15
30H	R/ESI	90H	IA32_PERF_GLOBAL_STATUS
38H	R/EDI	98H	Reserved
40H	R/EBP	A0H	Reserved
48H	R/ESP	A8H	Reserved
50H	R8	B0H	EventingRIP
58H	R9	B8H	Reserved

22.5.2.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table 22-66. "OffCore Response Event Encoding" lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

In the Silvermont microarchitecture, each MSR_OFFCORE_RSPx is shared by two processor cores.

Table 22-66. OffCore Response Event Encoding

Counter	Event code	UMask	Required Off-core Response MSR
PMC0-1	B7H	01H	MSR_OFFCORE_RSP0 (address 1A6H)
PMC0-1	B7H	02H	MSR_OFFCORE_RSP1 (address 1A7H)

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are shown in Figure 22-42 and Figure 22-43. Bits 15:0 specifies the request type of a transaction request to the uncore. Bits 30:16 specifies supplier information, bits 37:31 specifies snoop response information.

Additionally, MSR_OFFCORE_RSP0 provides bit 38 to enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously, see Section 22.5.2.3 for details.

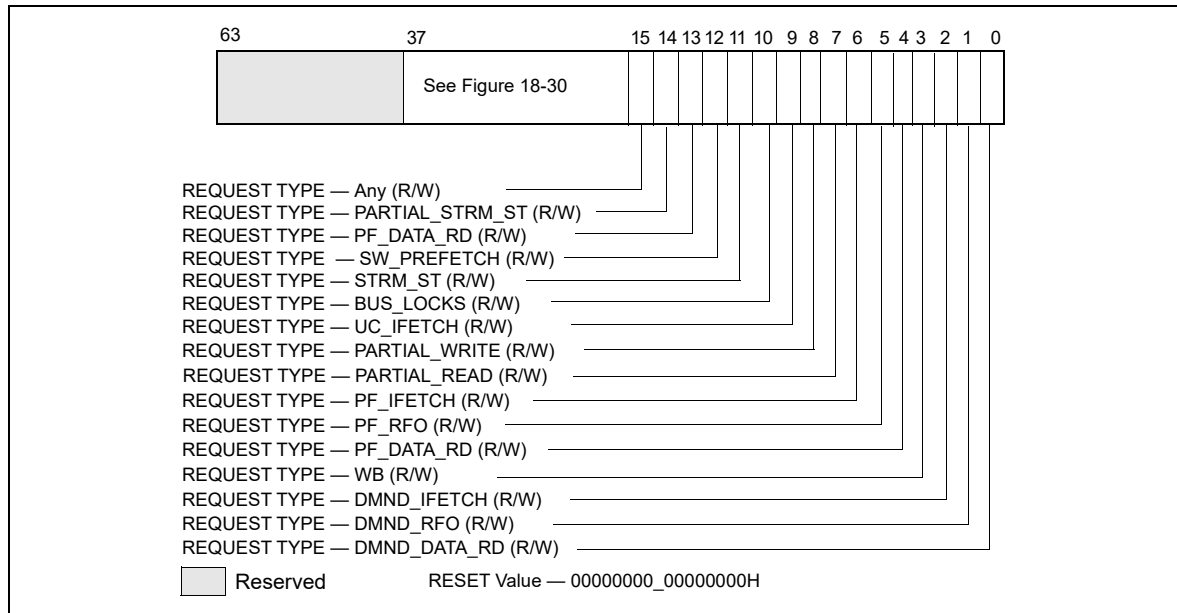


Figure 22-42. Request_Type Fields for MSR_OFFCORE_RSPx

Table 22-67. MSR_OFFCORE_RSPx Request_Type Field Definition

Bit Name	Offset	Description
DMND_DATA_RD	0	Counts the number of demand and DCU prefetch data reads of full and partial cachelines as well as demand data page table entry cacheline reads. Does not count L2 data read prefetches or instruction fetches.
DMND_RFO	1	Counts the number of demand and DCU prefetch reads for ownership (RFO) requests generated by a write to data cacheline. Does not count L2 RFO prefetches.
DMND_IFETCH	2	Counts the number of demand instruction cacheline reads and L1 instruction cacheline prefetches.
WB	3	Counts the number of writeback (modified to exclusive) transactions.
PF_DATA_RD	4	Counts the number of data cacheline reads generated by L2 prefetchers.
PF_RFO	5	Counts the number of RFO requests generated by L2 prefetchers.
PF_IFETCH	6	Counts the number of code reads generated by L2 prefetchers.
PARTIAL_READ	7	Counts the number of demand reads of partial cache lines (including UC and WC).
PARTIAL_WRITE	8	Counts the number of demand RFO requests to write to partial cache lines (includes UC, WT, and WP).
UC_IFETCH	9	Counts the number of UC instruction fetches.
BUS_LOCKS	10	Bus lock and split lock requests.
STRM_ST	11	Streaming store requests.
SW_PREFETCH	12	Counts software prefetch requests.
PF_DATA_RD	13	Counts DCU hardware prefetcher data read requests.
PARTIAL_STRM_ST	14	Streaming store requests.
ANY	15	Any request that crosses IDI, including I/O.

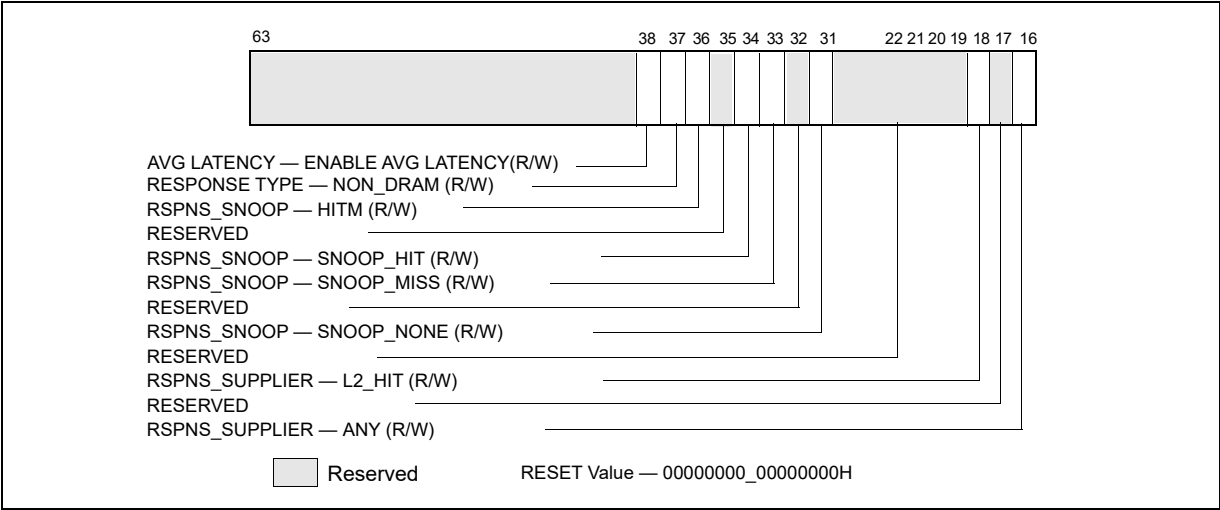


Figure 22-43. Response_Supplier and Snoop Info Fields for MSR_OFFCORE_RSPx

To properly program this extra register, software must set at least one request type bit (Table Table 22-67. "MSR_OFFCORE_RSPx Request_Type Field Definition") and a valid response type pattern (Table Table 22-68. "MSR_OFFCORE_RSP_x Response Supplier Info Field Definition", Table Table 22-69. "MSR_OFFCORE_RSPx Snoop Info Field Definition"). Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSPx allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 22-68. MSR_OFFCORE_RSP_x Response Supplier Info Field Definition

Subtype	Bit Name	Offset	Description
Common	ANY_RESPONSE	16	Catch all value for any response types.
Supplier Info	Reserved	17	Reserved
	L2_HIT	18	Cache reference hit L2 in either M/E/S states.
	Reserved	30:19	Reserved

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

ANY | [(‘OR’ of Supplier Info Bits) & (‘OR’ of Snoop Info Bits)]

If “ANY” bit is set, the supplier and snoop info bits are ignored.

Table 22-69. MSR_OFFCORE_RSPx Snoop Info Field Definition

Subtype	Bit Name	Offset	Description
Snoop Info	SNP_NONE	31	No details on snoop-related information.
	Reserved	32	Reserved
	SNOOP_MISS	33	Counts the number of snoop misses when L2 misses.
	SNOOP_HIT	34	Counts the number of snoops hit in the other module where no modified copies were found.

Table 22-69. MSR_OFFCORE_RSPx Snoop Info Field Definition (Contd.)

Subtype	Bit Name	Offset	Description
	Reserved	35	Reserved
	HITM	36	Counts the number of snoops hit in the other module where modified copies were found in other core's L1 cache.
	NON_DRAM	37	Target was non-DRAM system address. This includes MMIO transactions.
	AVG_LATENCY	38	Enable average latency measurement by counting weighted cycles of outstanding offcore requests of the request type specified in bits 15:0 and any response (bits 37:16 cleared to 0). This bit is available in MSR_OFFCORE_RESP0. The weighted cycles is accumulated in the specified programmable counter IA32_PMCx and the occurrence of specified requests are counted in the other programmable counter.

22.5.2.3 Average Offcore Request Latency Measurement

Average latency for offcore transactions can be determined by using both MSR_OFFCORE_RSP registers. Using two performance monitoring counters, program the two OFFCORE_RESPONSE event encodings into the corresponding IA32_PERFVTSELx MSRs. Count the weighted cycles via MSR_OFFCORE_RSP0 by programming a request type in MSR_OFFCORE_RSP0.[15:0] and setting MSR_OFFCORE_RSP0.OUTSTANDING[38] to 1, while setting the remaining bits to 0. Count the number of requests via MSR_OFFCORE_RSP1 by programming the same request type from MSR_OFFCORE_RSP0 into MSR_OFFCORE_RSP1[bit 15:0], and setting MSR_OFFCORE_RSP1.ANY_RESPONSE[16] = 1, while setting the remaining bits to 0. The average latency can be obtained by dividing the value of the IA32_PMCx register that counted weight cycles by the register that counted requests.

22.5.3 Performance Monitoring for Goldmont Microarchitecture

Intel Atom processors based on the Goldmont microarchitecture report architectural performance monitoring versionID = 4 (see Section 22.2.4) and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 4 capabilities are described in Section 22.2.4.

The bit fields (except bit 21) within each IA32_PERFVTSELx MSR are defined in Figure 22-6 and described in Section and Section 22.2.3. The Goldmont microarchitecture does not support Hyper-Threading and thus architectural and non-architectural performance monitoring events ignore the AnyThread qualification regardless of its setting in the IA32_PERFVTSELx MSR. However, Goldmont does not set the AnyThread deprecation bit (CPUID.0AH:EDX[15]).

The core PMU's capability is similar to that of the Silvermont microarchitecture described in Section 22.5.2, with some differences and enhancements summarized in Table Table 22-70. "Core PMU Comparison Between the Goldmont and Silvermont Microarchitectures".

Table 22-70. Core PMU Comparison Between the Goldmont and Silvermont Microarchitectures

Box	Goldmont Microarchitecture	Silvermont Microarchitecture	Comment
# of Fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	4	2	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:40, W:32	See Section 22.2.2.
Architectural Performance Monitoring version ID	4	3	Use CPUID to determine # of counters. See Section 22.2.1.

Table 22-70. Core PMU Comparison Between the Goldmont and Silvermont Microarchitectures

Box	Goldmont Microarchitecture	Silvermont Microarchitecture	Comment
PMI Overhead Mitigation	- Freeze_PerfMon_on_PMI with streamlined semantics. - Freeze_LBR_on_PMI with streamlined semantics for branch profiling.	- Freeze_PerfMon_on_PMI with legacy semantics. - Freeze_LBR_on_PMI with legacy semantics for branch profiling.	See Section 20.4.7. Legacy semantics not supported with version 4 or higher.
Counter and Buffer Overflow Status Management	- Query via IA32_PERF_GLOBAL_STATUS - Reset via IA32_PERF_GLOBAL_STATUS_RESET - Set via IA32_PERF_GLOBAL_STATUS_SET	- Query via IA32_PERF_GLOBAL_STATUS - Reset via IA32_PERF_GLOBAL_OVF_CTRL	See Section 22.2.4.
IA32_PERF_GLOBAL_STATUS Indicators of Overflow/Overhead/Interference	- Individual counter overflow - PEBS buffer overflow - ToPA buffer overflow - CTR_Frz, LBR_Frz	- Individual counter overflow - PEBS buffer overflow	See Section 22.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	- CTR_Frz, - LBR_Frz	No	See Section 22.2.4.1.
PerfMon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	No	See Section 22.2.4.3.
Processor Event Based Sampling (PEBS) Events	General-Purpose Counter 0 only. Supports all events (precise and non-precise). Precise events are listed in Table 22-71. "Precise Events Supported by the Goldmont Microarchitecture".	See Section 22.5.2.1.1. General-Purpose Counter 0 only. Only supports precise events (see Table 22-64. "PEBS Performance Events for the Silvermont Microarchitecture").	IA32_PMC0 only.
PEBS record format encoding	0011b	0010b	
Reduce skid PEBS	IA32_PMC0 only	No	
Data Address Profiling	Yes	No	
PEBS record layout	Table 22-72. "PEBS Record Format for the Goldmont Microarchitecture"; enhanced fields at offsets 90H- 98H; and TSC record field at COH.	Table 22-65. "PEBS Record Format for the Silvermont Microarchitecture".	
PEBS EventingIP	Yes	Yes	
Off-core Response Event	MSR 1A6H and 1A7H, each core has its own register.	MSR 1A6H and 1A7H, shared by a pair of cores.	Nehalem supports 1A6H only.

22.5.3.1 Processor Event Based Sampling (PEBS)

Processor event based sampling (PEBS) on the Goldmont microarchitecture is enhanced over prior generations with respect to sampling support of precise events and non-precise events. In the Goldmont microarchitecture, PEBS is supported using IA32_PMC0 for all events (see Section 20.4.9).

PEBS uses a debug store mechanism to store a set of architectural state information for the processor at the time the sample was generated.

Precise events work the same way on Goldmont microarchitecture as on the Silvermont microarchitecture. The record will be generated after an instruction that causes the event when the counter is already overflowed and will

capture the architectural state at this point (see Section 22.6.2.4 and Section 20.4.9). The eventingIP in the record will indicate the instruction that caused the event. The list of precise events supported in the Goldmont microarchitecture is shown in Table 22-71. "Precise Events Supported by the Goldmont Microarchitecture".

In the Goldmont microarchitecture, the PEBS facility also supports the use of non-precise events to record processor state information into PEBS records with the same format as with precise events.

However, a non-precise event may not be attributable to a particular retired instruction or the time of instruction execution. When the counter overflows, a PEBS record will be generated at the next opportunity. Consider the event ICACHE.HIT. When the counter overflows, the processor is fetching future instructions. The PEBS record will be generated at the next opportunity and capture the state at the processor's current retirement point. It is likely that the instruction fetch that caused the event to increment was beyond that current retirement point. Other examples of non-precise events are CPU_CLK_UNHALTED.CORE_P and HARDWARE_INTERRUPTS.RECEIVED. CPU_CLK_UNHALTED.CORE_P will increment each cycle that the processor is awake. When the counter overflows, there may be many instructions in various stages of execution. Additionally, zero, one or multiple instructions may be retired the cycle that the counter overflows. HARDWARE_INTERRUPTS.RECEIVED increments independent of any instructions being executed. For all non-precise events, the PEBS record will be generated at the next opportunity, after the counter has overflowed. The PEBS facility thus allows for identification of the instructions which were executing when the event overflowed.

After generating a record for a non-precise event, the PEBS facility reloads the counter and resumes execution, just as is done for precise events. Unlike interrupt-based sampling, which requires an interrupt service routine to collect the sample and reload the counter, the PEBS facility can collect samples even when interrupts are masked and without using NMI. Since a PEBS record is generated immediately when a counter for a non-precise event is enabled, it may also be generated after an overflow is set by an MSR write to IA32_PERF_GLOBAL_STATUS_SET.

Table 22-71. Precise Events Supported by the Goldmont Microarchitecture

Event Name	Event Select	Sub-event	UMask
LD_BLOCKS	03H	DATA_UNKNOWN	01H
		STORE_FORWARD	02H
		4K_ALIAS	04H
		UTLB_MISS	08H
		ALL_BLOCK	10H
MISALIGN_MEM_REF	13H	LOAD_PAGE_SPLIT	02H
		STORE_PAGE_SPLIT	04H
INST_RETIRED	C0H	ANY	00H
UOPS_RETIRED	C2H	ANY	00H
		LD_SPLITSMS	01H
BR_INST_RETIRED	C4H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		CALL	F9H
		REL_CALL	FDH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		FAR_BRANCH	BFH
		RETURN	F7H

Table 22-71. Precise Events Supported by the Goldmont Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		RETURN	F7H
MEM_UOPS_RETIRED	D0H	ALL_LOADS	81H
		ALL_STORES	82H
		ALL	83H
		DLTB_MISS_LOADS	11H
		DLTB_MISS_STORES	12H
		DLTB_MISS	13H
MEM_LOAD_UOPS_RETIRED	D1H	L1_HIT	01H
		L2_HIT	02H
		L1_MISS	08H
		L2_MISS	10H
		HITM	20H
		WCB_HIT	40H
		DRAM_HIT	80H

The PEBS record format supported by processors based on the Goldmont microarchitecture is shown in Table Table 22-72. "PEBS Record Format for the Goldmont Microarchitecture", and each field in the PEBS record is 64 bits long.

Table 22-72. PEBS Record Format for the Goldmont Microarchitecture

Byte Offset	Field	Byte Offset	Field
00H	R/EFLAGS	68H	R11
08H	R/EIP	70H	R12
10H	R/EAX	78H	R13
18H	R/EBX	80H	R14
20H	R/ECX	88H	R15
28H	R/EDX	90H	Applicable Counters
30H	R/ESI	98H	Data Linear Address
38H	R/EDI	A0H	Reserved
40H	R/EBP	A8H	Reserved
48H	R/ESP	B0H	EventingRIP
50H	R8	B8H	Reserved
58H	R9	C0H	TSC
60H	R10		

On Goldmont microarchitecture, all 64 bits of architectural registers are written into the PEBS record regardless of processor mode.

With PEBS record format encoding 0011b, offset 90H reports the “Applicable Counter” field, which indicates which counters actually requested generating a PEBS record. This allows software to correlate the PEBS record entry properly with the instruction that caused the event even when multiple counters are configured to record PEBS records and multiple bits are set in the field. Additionally, offset C0H captures a snapshot of the TSC that provides a time line annotation for each PEBS record entry.

22.5.3.1.1 PEBS Data Linear Address Profiling

Goldmont supports the Data Linear Address field introduced in Haswell. It does not support the Data Source Encoding or Latency Value fields that are also part of Data Address Profiling; those fields are present in the record but are reserved.

For Goldmont microarchitecture, the Data Linear Address field will record the linear address of memory accesses in the previous instruction (e.g., the one that triggered a precise event that caused the PEBS record to be generated). Goldmont microarchitecture may record a Data Linear Address for the instruction that caused the event even for events not related to memory accesses. This may differ from other microarchitectures.

22.5.3.1.2 Reduced Skid PEBS

Processors based on Goldmont Plus microarchitecture support the Reduced Skid PEBS feature described in Section 22.9.4 on the IA32_PMC0 counter. Although Extended PEBS adds support for generating PEBS records for precise events on additional general-purpose and fixed-function performance counters, those counters do not support the Reduced Skid PEBS feature.

22.5.3.1.3 Enhancements to IA32_PERF_GLOBAL_STATUS.OvfDSBuffer[62]

In addition to IA32_PERF_GLOBAL_STATUS.OvfDSBuffer[62] being set when PEBS_Index reaches the PEBS_Interrupt_Threshold, the bit is also set when PEBS_Index is out of bounds. That is, the bit will be set when PEBS_Index < PEBS_Buffer_Base or PEBS_Index > PEBS_Absolute_Maximum. Note that when an out of bound condition is encountered, the overflow bits in IA32_PERF_GLOBAL_STATUS will be cleared according to Applicable Counters, however the IA32_PMCx values will not be reloaded with the Reset values stored in the DS_AREA.

22.5.3.2 Offcore Response Event

Event number 0B7H support offcore response monitoring using an associated configuration MSR, MSR_OFFCORE_RSP0 (address 1A6H) in conjunction with UMASK value 01H or MSR_OFFCORE_RSP1 (address 1A7H) in conjunction with UMASK value 02H. Table Table 22-66. "OffCore Response Event Encoding" lists the event code, mask value and additional off-core configuration MSR that must be programmed to count off-core response events using IA32_PMCx.

The Goldmont microarchitecture provides unique pairs of MSR_OFFCORE_RSPx registers per core.

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as follows:

- Bits 15:0 specifies the request type of a transaction request to the uncore. This is described in Table Table 22-73. "MSR_OFFCORE_RSPx Request_Type Field Definition".
- Bits 30:16 specifies common supplier information or an L2 Hit, and is described in Table Table 22-68. "MSR_OFFCORE_RSP_x Response Supplier Info Field Definition".
- If L2 misses, then Bits 37:31 can be used to specify snoop response information and is described in Table Table 22-74. "MSR_OFFCORE_RSPx For L2 Miss and Outstanding Requests".
- For outstanding requests, bit 38 can enable measurement of average latency of specific type of offcore transaction requests using two programmable counter simultaneously; see Section 22.5.2.3 for details.

Table 22-73. MSR_OFFCORE_RSPx Request_Type Field Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts cacheline read requests due to demand reads (excludes prefetches).

Table 22-73. MSR_OFFCORE_RSPx Request_Type Field Definition (Contd.)

Bit Name	Offset	Description
DEMAND_RFO	1	Counts cacheline read for ownership (RFO) requests due to demand writes (excludes prefetches).
DEMAND_CODE_RD	2	Counts demand instruction cacheline and I-side prefetch requests that miss the instruction cache.
COREWB	3	Counts writeback transactions caused by L1 or L2 cache evictions.
PF_L2_DATA_RD	4	Counts data cacheline reads generated by hardware L2 cache prefetcher.
PF_L2_RFO	5	Counts reads for ownership (RFO) requests generated by L2 prefetcher.
Reserved	6	Reserved.
PARTIAL_READS	7	Counts demand data partial reads, including data in uncacheable (UC) or uncacheable (WC) write combining memory types.
PARTIAL_WRITES	8	Counts partial writes, including uncacheable (UC), write through (WT) and write protected (WP) memory type writes.
UC_CODE_READS	9	Counts code reads in uncacheable (UC) memory region.
BUS_LOCKS	10	Counts bus lock and split lock requests.
FULL_STREAMING_STORES	11	Counts full cacheline writes due to streaming stores.
SW_PREFETCH	12	Counts cacheline requests due to software prefetch instructions.
PF_L1_DATA_RD	13	Counts data cacheline reads generated by hardware L1 data cache prefetcher.
PARTIAL_STREAMING_STORES	14	Counts partial cacheline writes due to streaming stores.
ANY_REQUEST	15	Counts requests to the uncore subsystem.

To properly program this extra register, software must set at least one request type bit (Table Table 22-67. "MSR_OFFCORE_RSPx Request_Type Field Definition") and a valid response type pattern (either Table Table 22-68. "MSR_OFFCORE_RSP_x Response Supplier Info Field Definition" or Table Table 22-74. "MSR_OFFCORE_RSPx For L2 Miss and Outstanding Requests"). Otherwise, the event count reported will be zero. It is permissible and useful to set multiple request and response type bits in order to obtain various classes of off-core response events. Although MSR_OFFCORE_RSPx allow an agent software to program numerous combinations that meet the above guideline, not all combinations produce meaningful data.

Table 22-74. MSR_OFFCORE_RSPx For L2 Miss and Outstanding Requests

Subtype	Bit Name	Offset	Description
L2_MISS (Snoop Info)	Reserved	32:31	Reserved
	L2_MISS.SNOOP_MISS_0 R_NO_SNOOP_NEEDED	33	A true miss to this module, for which a snoop request missed the other module or no snoop was performed/needed.
	L2_MISS.HIT_OTHER_CO RE_NO_FWD	34	A snoop hit in the other processor module, but no data forwarding is required.
	Reserved	35	Reserved
	L2_MISS.HITM_OTHER_C ORE	36	Counts the number of snoops hit in the other module or other core's L1 where modified copies were found.
	L2_MISS.NON_DRAM	37	Target was a non-DRAM system address. This includes MMIO transactions.
Outstanding requests ¹	OUTSTANDING	38	Counts weighted cycles of outstanding offcore requests of the request type specified in bits 15:0, from the time the XQ receives the request and any response is received. Bits 37:16 must be set to 0. This bit is only available in MSR_OFFCORE_RESP0.

NOTES:

1. See Section 22.5.2.3, “Average Offcore Request Latency Measurement,” for details on how to use this bit to extract average latency.

To specify a complete offcore response filter, software must properly program bits in the request and response type fields. A valid request type must have at least one bit set in the non-reserved bits of 15:0. A valid response type must be a non-zero value of the following expression:

Any_Response Bit | L2 Hit | ‘OR’ of Snoop Info Bits | Outstanding Bit

22.5.3.3 Average Offcore Request Latency Measurement

In Goldmont microarchitecture, measurement of average latency of offcore transaction requests is the same as described in Section 22.5.2.3.

22.5.4 Performance Monitoring for Goldmont Plus Microarchitecture

Intel Atom processors based on the Goldmont Plus microarchitecture report architectural performance monitoring versionID = 4 and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 4 capabilities are described in Section 22.2.4.

Goldmont Plus performance monitoring capabilities are similar to Goldmont capabilities. The differences are in specific events and in which counters support PEBS. Goldmont Plus introduces the ability for fixed performance monitoring counters to generate PEBS records.

Goldmont Plus will set the AnyThread deprecation CPUID bit (CPUID.0AH:EDX[15]) to indicate that the Any-Thread bits in IA32_PERFVTSELx and IA32_FIXED_CTR_CTRL have no effect.

The core PMU's capability is similar to that of the Goldmont microarchitecture described in Section 22.6.3, with some differences and enhancements summarized in Table 22-75.

Table 22-75. Core PMU Comparison Between the Goldmont Plus and Goldmont Microarchitectures

Box	Goldmont Plus Microarchitecture	Goldmont Microarchitecture	Comment
# of Fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	4	4	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	No change.
Architectural Performance Monitoring version ID	4	4	No change.
Processor Event Based Sampling (PEBS) Events	All General-Purpose and Fixed counters. Each General-Purpose counter supports all events (precise and non-precise).	General-Purpose Counter 0 only. Supports all events (precise and non-precise). Precise events are listed in Table 22-71. “Precise Events Supported by the Goldmont Microarchitecture”.	Goldmont Plus supports PEBS on all counters.
PEBS record format encoding	0011b	0011b	No change.

22.5.4.1 Extended PEBS

The PEBS facility in Goldmont Plus microarchitecture provides a number of enhancements relative to PEBS in processors from previous generations. Enhancement of PEBS facility with the Extended PEBS feature are described in detail in section 18.9.

22.5.5 Performance Monitoring for Tremont Microarchitecture

Intel Atom processors based on the Tremont microarchitecture report architectural performance monitoring versionID = 5 and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 5 capabilities are described in Section 22.2.5.

Tremont performance monitoring capabilities are similar to Goldmont Plus capabilities, with the following extensions:

- Support for Adaptive PEBS.
- Support for PEBS output to Intel® Processor Trace.
- Precise Distribution support on Fixed Counter0.
- Compatibility enhancements to off-core response MSRs, MSR_OFFCORE_RSPx.

The differences and enhancements between Tremont microarchitecture and Goldmont Plus microarchitecture are summarized in Table 22-76.

Table 22-76. Core PMU Comparison Between the Tremont and Goldmont Plus Microarchitectures

Box	Tremont Microarchitecture	Goldmont Plus Microarchitecture	Comment
# of fixed counters per core	3	3	Use CPUID to determine # of counters. See Section 22.2.1.
# of general-purpose counters per core	4	4	Use CPUID to determine # of counters. See Section 22.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	No change. See Section 22.2.2.
Architectural Performance Monitoring version ID	5	4	
PEBS record format encoding	0100b	0011b	See Section 22.6.2.4.2.
Reduce skid PEBS	IA32_PMC0 and IA32_FIXED_CTR0	IA32_PMC0 only	
Extended PEBS	Yes	Yes	See Section 22.5.4.1.
Adaptive PEBS	Yes	No	See Section 22.9.2.
PEBS output	DS Save Area or Intel® Processor Trace	DS Save Area only	See Section 22.5.5.2.1.
PEBS record layout	See Section 22.9.2.3 for output to DS, Section 22.5.5.2.2 for output to Intel PT.	Table 22-72; enhanced fields at offsets 90H- 98H; and TSC record field at COH.	
Off-core Response Event	MSR 1A6H and 1A7H, each core has its own register, extended request and response types.	MSR 1A6H and 1A7H, each core has its own register.	

22.5.5.1 Adaptive PEBS

The PEBS record format and configuration interface has changed versus Goldmont Plus, as the Tremont microarchitecture includes support for the configurable Adaptive PEBS records; see Section 22.9.2.

22.5.5.2 PEBS output to Intel® Processor Trace

Intel Atom processors based on the Tremont microarchitecture introduce the following Precise Event-Based Sampling (PEBS) extensions:

- A mechanism to direct PEBS output into the Intel® Processor Trace (Intel® PT) output stream. In this scenario, the PEBS record is written in packetized form, in order to co-exist with other Intel PT trace data.
- New Performance Monitoring counter reload MSRs, which are used by PEBS in place of the counter reload values stored in the DS Management area when PEBS output is directed into the Intel PT output stream.

Processors that indicate support for Intel PT by setting CPUID.07H.00H:EBX[25]=1, and set the new IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16] bit, support these extensions.

22.5.5.2.1 PEBS Configuration

PEBS output to Intel Processor Trace includes support for two new fields in IA32_PEBS_ENABLE.

Table 22-77. New Fields in IA32_PEBS_ENABLE

Field	Description
PMI_AFTER_EACH_RECORD[60]	Pend a PerfMon Interrupt (PMI) after each PEBS event.
PEBS_OUTPUT[62:61]	Specifies PEBS output destination. Encodings: 00B: DS Save Area. Matches legacy PEBS behavior, output location defined by IA32_DS_AREA. 01B: Intel PT trace output. 10B: Reserved. 11B: Reserved.

When PEBS_OUTPUT is set to 01B, the DS Management Area is not used and need not be configured. Instead, the output mechanism is configured through IA32_RTIT_CTL and other Intel PT MSRs, while counter reload values are configured in the MSR_RELOAD_PMCx MSRs. Details on configuring Intel PT can be found in Section 36.2.7.

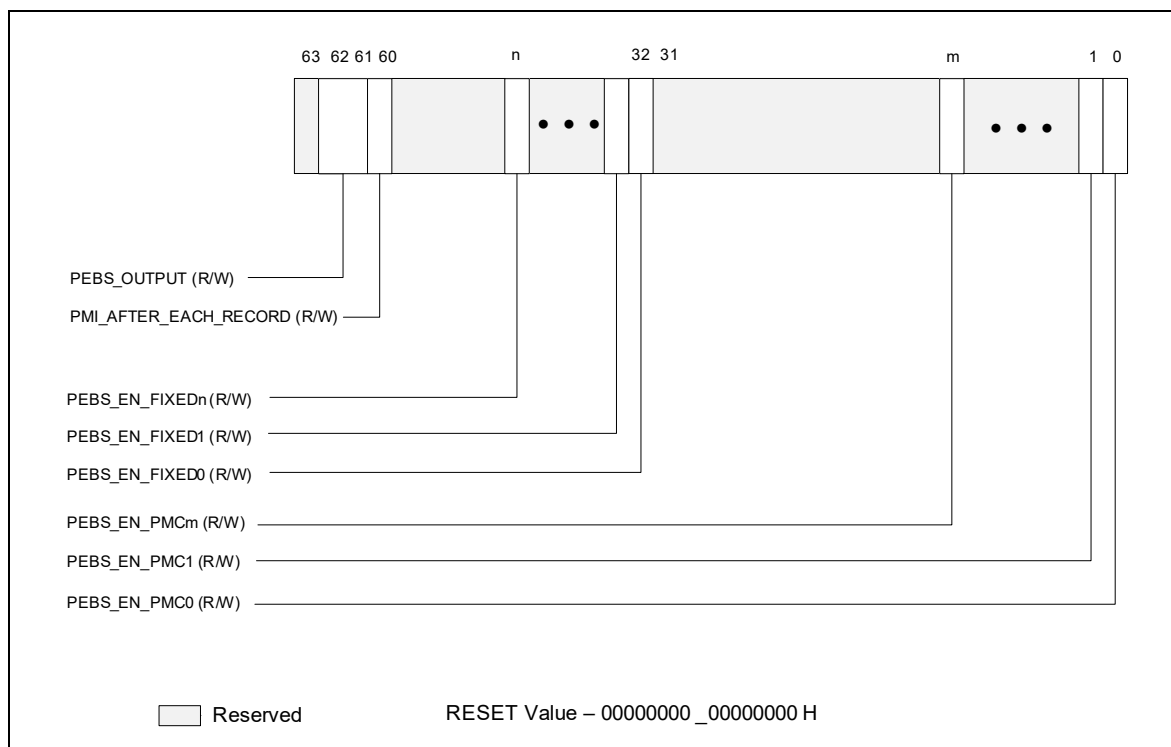


Figure 22-44. IA32_PEBS_ENABLE MSR with PEBS Output to Intel® Processor Trace

22.5.5.2.2 PEBS Record Format in Intel® Processor Trace

The format of the PEBS record changes when output to Intel PT, as the PEBS state is packetized. Each PEBS grouping is emitted as a Block Begin (BBP) and following Block Item (BIP) packets. A PEBS grouping ends when either a new PEBS grouping begins (indicated by a BBP packet) or a Block End (BEP) packet is encountered. See Section 36.4.1.1 for details of these Intel PT packets.

Because the packet headers describe the state held in the packet payload, PEBS state ordering is not fixed. PEBS state groupings may be emitted in any order, and the PEBS state elements within those groupings may be emitted in any order. Further, there is no packet that provides indication of “Record Format” or “Record Size”.

If Intel PT tracing is not enabled (IA32_RTIT_STATUS.TriggerEn=0), any PEBS records triggered will be dropped. PEBS packets do not depend on ContextEn or FilterEn in IA32_RTIT_STATUS, any filtering of PEBS must be enabled from within the PerfMon configuration. Counter reload will occur in all scenarios where PEBS is triggered, regardless of TriggerEn.

The PEBS threshold mechanism for generating PerfMon Interrupts (PMIs) is not available in this mode. However, there exist other means to generate PMIs based on PEBS output. When the Intel PT ToPA output mechanism is chosen, a PMI can optionally be pended when a ToPA region is filled; see Section 36.2.7.2 for details. Further, software can opt to generate a PMI on each PEBS record by setting the new IA32_PEBS_ENABLE.PMI_AFTER_EACH_RECORD[60] bit.

The IA32_PERF_GLOBAL_STATUS.OvfDSBuffer bit will not be set in this mode.

22.5.5.2.3 PEBS Counter Reload

When PEBS output is directed into Intel PT (IA32_PEBS_ENABLE.PEBS_OUTPUT = 01B), new MSR_RELOAD_PMCx MSRs are used by the PEBS routine to reload PerfMon counters. The value from the associated reload MSR will be loaded to the appropriate counter on each PEBS event.

22.5.5.3 Precise Distribution Support on Fixed Counter 0

The Tremont microarchitecture supports the PDIR (Precise Distribution of Retired Instructions) facility, as described in Section 22.3.4.4.4, on Fixed Counter 0. Fixed Counter 0 counts the INST_RETIRED.ALL event. PEBS skid for Fixed Counter 0 will be precisely one instruction.

This is in addition to the reduced skid PEBS behavior on IA32_PMC0; see Section 22.5.3.1.2.

22.5.5.4 Compatibility Enhancements to Offcore Response MSRs

The Off-core Response facility is similar to that described in Section 22.5.3.2.

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as shown below. RequestType bits are defined in Table 22-78, ResponseType bits in Table 22-79, and SnoopInfo bits in Table 22-80.

Table 22-78. MSR_OFFCORE_RSPx Request Type Definition

Bit Name	Offset	Description
DEMAND_DATA_RD	0	Counts demand data reads.
DEMAND_RFO	1	Counts all demand reads for ownership (RFO) requests and software based prefetches for exclusive ownership (prefetchw).
DEMAND_CODE_RD	2	Counts demand instruction fetches and L1 instruction cache prefetches.
COREWB_M	3	Counts modified write backs from L1 and L2.
HWPf_L2_DATA_RD	4	Counts prefetch (that bring data to L2) data reads.
HWPf_L2_RFO	5	Counts all prefetch (that bring data to L2) RFOs.
HWPf_L2_CODE_RD	6	Counts all prefetch (that bring data to L2 only) code reads.
Reserved	9:7	Reserved.

Table 22-78. MSR_OFFCORE_RSPx Request Type Definition (Contd.)

Bit Name	Offset	Description
HWPf_L1D_AND_SWPF	10	Counts L1 data cache hardware prefetch requests, read for ownership prefetch requests and software prefetch requests (except prefetchw).
STREAMING_WR	11	Counts all streaming stores.
COREWB_NONM	12	Counts non-modified write backs from L2.
Reserved	14:13	Reserved.
OTHER	15	Counts miscellaneous requests, such as I/O accesses that have any response type.
UC_RD	44	Counts uncached memory reads (PRd, UCRdF).
UC_WR	45	Counts uncached memory writes (WiL).
PARTIAL_STREAMING_WR	46	Counts partial (less than 64 byte) streaming stores (WCiL).
FULL_STREAMING_WR	47	Counts full, 64 byte streaming stores (WCiLF).
L1WB_M	48	Counts modified WriteBacks from L1 that miss the L2.
L2WB_M	49	Counts modified WriteBacks from L2.

Table 22-79. MSR_OFFCORE_RSPx Response Type Definition

Bit Name	Offset	Description
ANY_RESPONSE	16	Catch all value for any response types.
L3_HIT_M	18	LLC/L3 Hit - M-state.
L3_HIT_E	19	LLC/L3 Hit - E-state.
L3_HIT_S	20	LLC/L3 Hit - S-state.
L3_HIT_F	21	LLC/L3 Hit - I-state.
LOCAL_DRAM	26	LLC/L3 Miss, DRAM Hit.
OUTSTANDING	63	Average latency of outstanding requests with the other counter counting number of occurrences; can also can be used to count occupancy.

Table 22-80. MSR_OFFCORE_RSPx Snoop Info Definition

Bit Name	Offset	Description
SNOOP_NONE	31	None of the cores were snooped. ▪ LLC miss and Dram data returned directly to the core.
SNOOP_NOT_NEEDED	32	No snoop needed to satisfy the request. ▪ LLC hit and CV bit(s) (core valid) was not set. ▪ LLC miss and Dram data returned directly to the core.
SNOOP_MISS	33	A snoop was sent but missed. ▪ LLC hit and CV bit(s) was set but snoop missed (silent data drop in core), data returned from LLC. ▪ LLC miss and Dram data returned directly to the core.
SNOOP_HIT_NO_FWD	34	A snoop was sent but no data forward. ▪ LLC hit and CV bit(s) was set but no data forward from the core, data returned from LLC. ▪ LLC miss and Dram data returned directly to the core.

Table 22-80. MSR_OFFCORE_RSPx Snoop Info Definition

Bit Name	Offset	Description
SNOOP_HIT_WITH_FWD	35	A snoop was sent and non-modified data was forward. ▪ LLC hit and CV bit(s) was set, non-modified data was forward from core.
SNOOP_HITM	36	A snoop was sent and modified data was forward. ▪ LLC hit E or M and the CV bit(s) was set, modified data was forward from core.
NON_DRAM_BIT	37	Target was non-DRAM system address, MMIO access. ▪ LLC miss and Non-Dram data returned.

The Off-core Response capability behaves as follows:

- To specify a complete offcore response filter, software must properly program at least one RequestType and one ResponseType. A valid request type must have at least one bit set in the non-reserved bits of 15:0 or 49:44. A valid response type must be a non-zero value of one the following expressions:
 - Read requests:
Any_Response Bit | ('OR' of Supplier Info Bits) 'AND' ('OR' of Snoop Info Bits) | Outstanding Bit
 - Write requests:
Any_Response Bit | ('OR' of Supplier Info Bits) | Outstanding Bit
- When the ANY_RESPONSE bit in the ResponseType is set, all other response type bits will be ignored.
- True Demand Cacheable Loads include neither L1 Prefetches nor Software Prefetches.
- Bits 15:0 and Bits 49:44 specifies the request type of a transaction request to the uncore. This is described in Table 22-78.
- Bits 30:16 specifies common supplier information.
- "Outstanding Requests" (bit 63) is only available on MSR_OFFCORE_RSP0; a #GP fault will occur if software attempts to write a 1 to this bit in MSR_OFFCORE_RSP1. It is mutually exclusive with any ResponseType. Software must guarantee that all other ResponseType bits are set to 0 when the "Outstanding Requests" bit is set.
- "Outstanding Requests" bit 63 can enable measurement of the average latency of a specific type of off-core transaction; two programmable counters must be used simultaneously and the RequestType programming for MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 must be the same when using this Average Latency feature. See Section 22.5.2.3 for further details.

22.6 PERFORMANCE MONITORING (LEGACY INTEL PROCESSORS)

22.6.1 Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)

In Intel Core Solo and Intel Core Duo processors, non-architectural performance monitoring events are programmed using the same facilities (see Figure 22-1) used for architectural performance events.

Non-architectural performance events use event select values that are model-specific. Event mask (Umask) values are also specific to event logic units. Some microarchitectural conditions detectable by a Umask value may have specificity related to processor topology (see Section 11.6, "Detecting Hardware Multi-Threading Support and Topology," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A). As a result, the unit mask field (for example, IA32_PERFEVTSELx[bits 15:8]) may contain sub-fields that specify topology information of processor cores.

The sub-field layout within the Umask field may support two-bit encoding that qualifies the relationship between a microarchitectural condition and the originating core. This data is shown in Table 22-81. The two-bit encoding for core-specificity is only supported for a subset of Umask values (see: <https://perfmon-events.intel.com/>) and for Intel Core Duo processors. Such events are referred to as core-specific events.

Table 22-81. Core Specificity Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 15:14 Encoding	Description
11B	All cores
10B	Reserved
01B	This core
00B	Reserved

Some microarchitectural conditions allow detection specificity only at the boundary of physical processors. Some bus events belong to this category, providing specificity between the originating physical processor (a bus agent) versus other agents on the bus. Sub-field encoding for agent specificity is shown in Table Table 22-82. "Agent Specificity Encoding within a Non-Architectural Umask".

Table 22-82. Agent Specificity Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 13 Encoding	Description
0	This agent
1	Include all agents

Some microarchitectural conditions are detectable only from the originating core. In such cases, unit mask does not support core-specificity or agent-specificity encodings. These are referred to as core-only conditions.

Some microarchitectural conditions allow detection specificity that includes or excludes the action of hardware prefetches. A two-bit encoding may be supported to qualify hardware prefetch actions. Typically, this applies only to some L2 or bus events. The sub-field encoding for hardware prefetch qualification is shown in Table Table 22-83. "HW Prefetch Qualification Encoding within a Non-Architectural Umask".

Table 22-83. HW Prefetch Qualification Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 13:12 Encoding	Description
11B	All inclusive
10B	Reserved
01B	Hardware prefetch only
00B	Exclude hardware prefetch

Some performance events may (a) support none of the three event-specific qualification encodings (b) may support core-specificity and agent specificity simultaneously (c) or may support core-specificity and hardware prefetch qualification simultaneously. Agent-specificity and hardware prefetch qualification are mutually exclusive.

In addition, some L2 events permit qualifications that distinguish cache coherent states. The sub-field definition for cache coherency state qualification is shown in Table Table 22-84. "MESI Qualification Definitions within a Non-Architectural Umask". If no bits in the MESI qualification sub-field are set for an event that requires setting MESI qualification bits, the event count will not increment.

Table 22-84. MESI Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 11:8	Description
Bit 11	Counts modified state
Bit 10	Counts exclusive state

Table 22-84. MESI Qualification Definitions within a Non-Architectural Umask

IA32_PERFVTSELx MSRs	
Bit Position 11:8	Description
Bit 9	Counts shared state
Bit 8	Counts Invalid state

22.6.2 Performance Monitoring (Processors Based on Intel® Core™ Microarchitecture)

In addition to architectural performance monitoring, processors based on the Intel Core microarchitecture support non-architectural performance monitoring events.

Architectural performance events can be collected using general-purpose performance counters. Non-architectural performance events can be collected using general-purpose performance counters (coupled with two IA32_PERFVTSELx MSRs for detailed event configurations), or fixed-function performance counters (see Section 22.6.2.1). IA32_PERFVTSELx MSRs are architectural; their layout is shown in Figure 22-1. Starting with Intel Core 2 processor T 7700, fixed-function performance counters and associated counter control and status MSR becomes part of architectural performance monitoring version 2 facilities (see also Section 22.2.2).

Non-architectural performance events in processors based on Intel Core microarchitecture use event select values that are model-specific. Valid event mask (Umask) bits can be found at: <https://perfmon-events.intel.com/>. The UMASK field may contain sub-fields identical to those listed in Table Table 22-81. "Core Specificity Encoding within a Non-Architectural Umask", Table Table 22-82. "Agent Specificity Encoding within a Non-Architectural Umask", Table Table 22-83. "HW Prefetch Qualification Encoding within a Non-Architectural Umask", and Table Table 22-84. "MESI Qualification Definitions within a Non-Architectural Umask". One or more of these sub-fields may apply to specific events on an event-by-event basis.

In addition, the UMASK field may also contain a sub-field that allows detection specificity related to snoop responses. Bits of the snoop response qualification sub-field are defined in Table Table 22-85. "Bus Snoop Qualification Definitions within a Non-Architectural Umask".

Table 22-85. Bus Snoop Qualification Definitions within a Non-Architectural Umask

IA32_PERFVTSELx MSRs	
Bit Position 11:8	Description
Bit 11	HITM response
Bit 10	Reserved
Bit 9	HIT response
Bit 8	CLEAN response

There are also non-architectural events that support qualification of different types of snoop operation. The corresponding bit field for snoop type qualification are listed in Table 22-86.

Table 22-86. Snoop Type Qualification Definitions within a Non-Architectural Umask

IA32_PERFVTSELx MSRs	
Bit Position 9:8	Description
Bit 9	CMP2I snoops
Bit 8	CMP2S snoops

No more than one sub-field of MESI, snoop response, and snoop type qualification sub-fields can be supported in a performance event.

NOTE

Software must write known values to the performance counters prior to enabling the counters. The content of general-purpose counters and fixed-function counters are undefined after INIT or RESET.

22.6.2.1 Fixed-function Performance Counters

Processors based on Intel Core microarchitecture provide three fixed-function performance counters. Bits beyond the width of the fixed counter are reserved and must be written as zeros. Model-specific fixed-function performance counters on processors that support Architectural PerfMon version 1 are 40 bits wide.

Each of the fixed-function counter is dedicated to count a pre-defined performance monitoring events. See Table 22-1 for details of the PMC addresses and what these events count.

Programming the fixed-function performance counters does not involve any of the IA32_PERFEVTSELx MSRs, and does not require specifying any event masks. Instead, the MSR IA32_FIXED_CTR_CTRL provides multiple sets of 4-bit fields; each 4-bit field controls the operation of a fixed-function performance counter (PMC). See Figures 22-45. Two sub-fields are defined for each control. See Figure 22-45; bit fields are:

- **Enable field (low 2 bits in each 4-bit control)** — When bit 0 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment when the target condition associated with the architecture performance event occurs at ring 0.

When bit 1 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment when the target condition associated with the architecture performance event occurs at ring greater than 0.

Writing 0 to both bits stops the performance counter. Writing 11B causes the counter to increment irrespective of privilege levels.

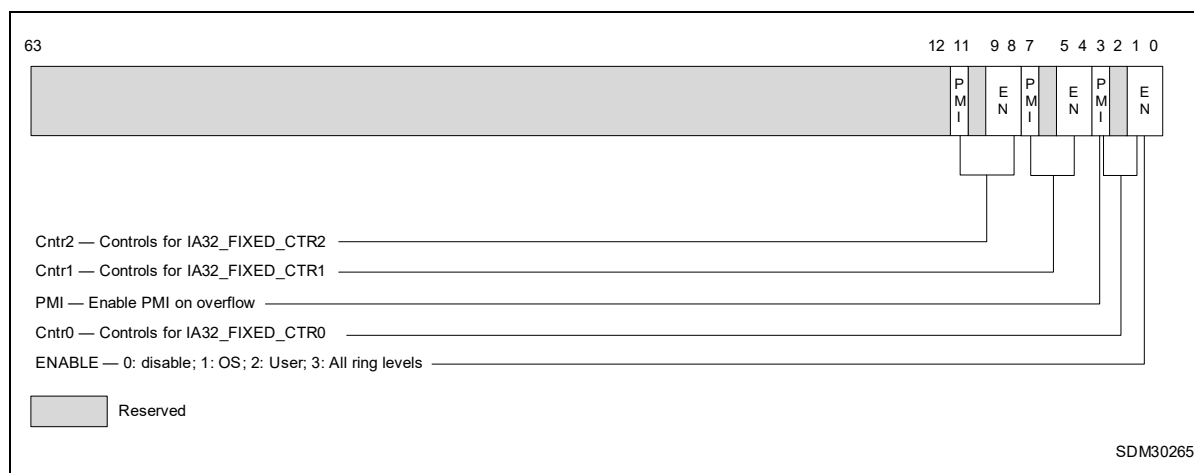


Figure 22-45. Layout of IA32_FIXED_CTR_CTRL MSR

- **PMI field (fourth bit in each 4-bit control)** — When set, the logical processor generates an exception through its local APIC on overflow condition of the respective fixed-function counter.

22.6.2.2 Global Counter Control Facilities

Processors based on Intel Core microarchitecture provides simplified performance counter control that simplifies the most frequent operations in programming performance events, i.e., enabling/disabling event counting and checking the status of counter overflows. This is done by the following three MSRs:

- MSR_PERF_GLOBAL_CTRL enables/disables event counting for all or any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.

- MSR_PERF_GLOBAL_STATUS allows software to query counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single RDMSR.
- MSR_PERF_GLOBAL_OVF_CTRL allows software to clear counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.

MSR_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting in each performance counter (see Figure 22-46). Each enable bit in MSR_PERF_GLOBAL_CTRL is ANDed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the ANDed results is true; counting is disabled when the result is false.

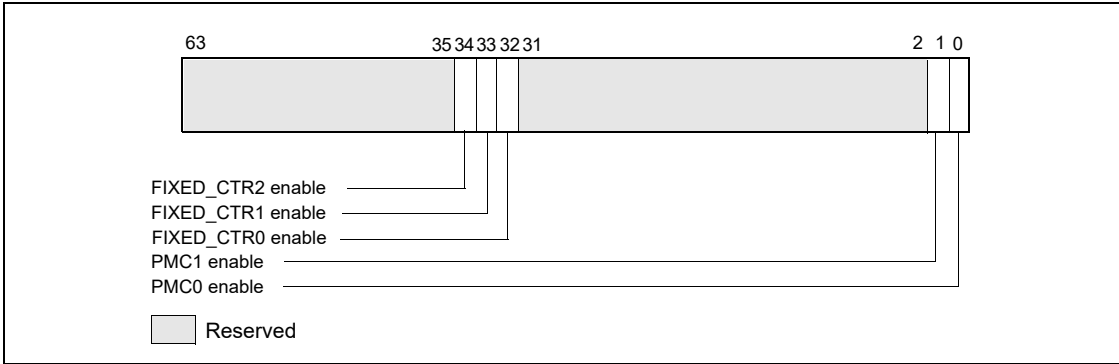


Figure 22-46. Layout of MSR_PERF_GLOBAL_CTRL MSR

MSR_PERF_GLOBAL_STATUS MSR provides single-bit status used by software to query the overflow condition of each performance counter. MSR_PERF_GLOBAL_STATUS[bit 62] indicates overflow conditions of the DS area data buffer. MSR_PERF_GLOBAL_STATUS[bit 63] provides a CondChgd bit to indicate changes to the state of performance monitoring hardware (see Figure 22-47). A value of 1 in bits 34:32, 1, 0 indicates an overflow condition has occurred in the associated counter.

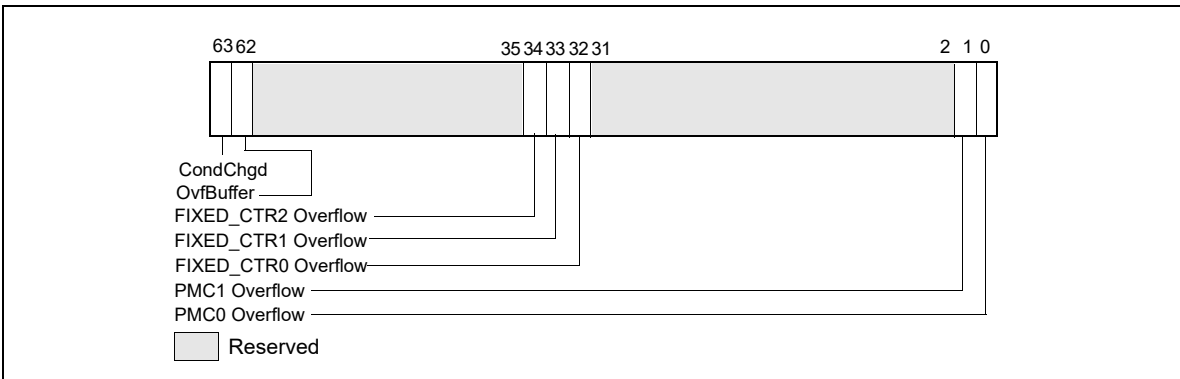


Figure 22-47. Layout of MSR_PERF_GLOBAL_STATUS MSR

When a performance counter is configured for PEBS, an overflow condition in the counter will arm PEBS. On the subsequent event following overflow, the processor will generate a PEBS event. On a PEBS event, the processor will perform bounds checks based on the parameters defined in the DS Save Area (see Section 20.4.9). Upon successful bounds checks, the processor will store the data record in the defined buffer area, clear the counter overflow status, and reload the counter. If the bounds checks fail, the PEBS will be skipped entirely. In the event that the PEBS buffer fills up, the processor will set the OvfBuffer bit in MSR_PERF_GLOBAL_STATUS.

MSR_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow the indicators for general-purpose or fixed-function counters via a single WRMSR (see Figure 22-48). Clear overflow indications when:

- Setting up new values in the event select and/or UMASK field for counting or interrupt-based event sampling.
- Reloading counter values to continue collecting next sample.
- Disabling event counting or interrupt-based event sampling.

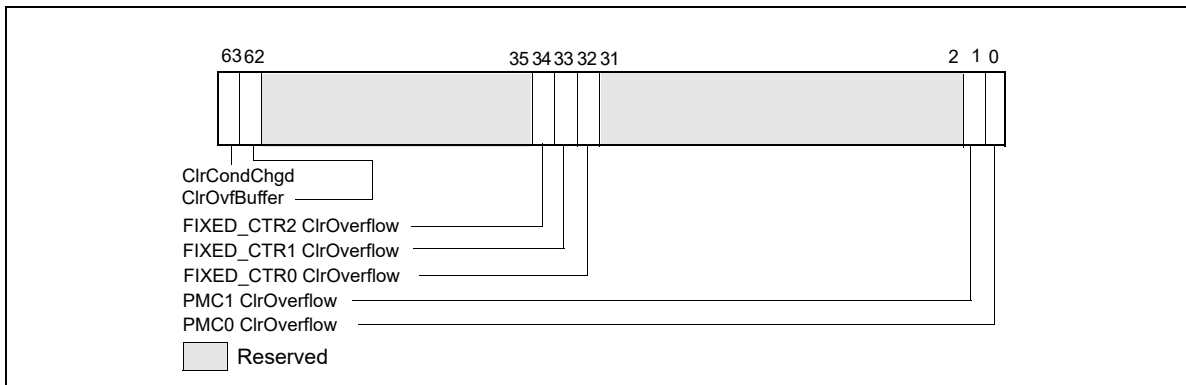


Figure 22-48. Layout of MSR_PERF_GLOBAL_OVF_CTRL MSR

22.6.2.3 At-Retirement Events

Many non-architectural performance events are impacted by the speculative nature of out-of-order execution. A subset of non-architectural performance events on processors based on Intel Core microarchitecture are enhanced with a tagging mechanism (similar to that found in Intel NetBurst® microarchitecture) that exclude contributions that arise from speculative execution. The at-retirement events available in processors based on Intel Core microarchitecture does not require special MSR programming control (see Section 22.6.3.6, "At-Retirement Counting"), but is limited to IA32_PMC0. See Table 22-87. "At-Retirement Performance Events for Intel Core Microarchitecture" for a list of events available to processors based on Intel Core microarchitecture.

Table 22-87. At-Retirement Performance Events for Intel Core Microarchitecture

Event Name	UMask	Event Select
ITLB_MISS_RETIRED	00H	C9H
MEM_LOAD_RETIRED.L1D_MISS	01H	CBH
MEM_LOAD_RETIRED.L1D_LINE_MISS	02H	CBH
MEM_LOAD_RETIRED.L2_MISS	04H	CBH
MEM_LOAD_RETIRED.L2_LINE_MISS	08H	CBH
MEM_LOAD_RETIRED.DTLB_MISS	10H	CBH

22.6.2.4 Processor Event Based Sampling (PEBS)

Processors based on Intel Core microarchitecture also support processor event based sampling (PEBS). This feature was introduced by processors based on Intel NetBurst microarchitecture.

PEBS uses a debug store mechanism and a performance monitoring interrupt to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 22.6.2.4.2 and Section 20.4.9).

In cases where the same instruction causes BTS and PEBS to be activated, PEBS is processed before BTS are processed. The PMI request is held until the processor completes processing of PEBS and BTS.

For processors based on Intel Core microarchitecture, precise events that can be used with PEBS are listed in Table 22-88. "PEBS Performance Events for Intel Core Microarchitecture". The procedure for detecting availability of PEBS is the same as described in Section 22.6.3.8.1.

Table 22-88. PEBS Performance Events for Intel Core Microarchitecture

Event Name	UMask	Event Select
INSTR_RETIRED.ANY_P	00H	C0H
X87_OPS_RETIRED.ANY	FEH	C1H
BR_INST_RETIRED.MISPRED	00H	C5H
SIMD_INST_RETIRED.ANY	1FH	C7H
MEM_LOAD_RETIRED.L1D_MISS	01H	CBH
MEM_LOAD_RETIRED.L1D_LINE_MISS	02H	CBH
MEM_LOAD_RETIRED.L2_MISS	04H	CBH
MEM_LOAD_RETIRED.L2_LINE_MISS	08H	CBH
MEM_LOAD_RETIRED.DTLB_MISS	10H	CBH

22.6.2.4.1 Setting up the PEBS Buffer

For processors based on Intel Core microarchitecture, PEBS is available using IA32_PMC0 only. Use the following procedure to set up the processor and IA32_PMC0 counter for PEBS:

1. Set up the precise event buffering facilities. Place values in the precise event buffer base, precise event index, precise event absolute maximum, precise event interrupt threshold, and precise event counter reset fields of the DS buffer management area. In processors based on Intel Core microarchitecture, PEBS records consist of 64-bit address entries. See Figure 20-8 to set up the precise event records buffer in memory.
2. Enable PEBS. Set the Enable PEBS on PMC0 flag (bit 0) in IA32_PEBS_ENABLE MSR.
3. Set up the IA32_PMC0 performance counter and IA32_PERFEVTSEL0 for an event listed in Table Table 22-88. "PEBS Performance Events for Intel Core Microarchitecture".

22.6.2.4.2 PEBS Record Format

The PEBS record format may be extended across different processor implementations. The IA32_PERF_CAPABILITIES MSR defines a mechanism for software to handle the evolution of PEBS record format in processors that support architectural performance monitoring with version ID equals 2 or higher. The bit fields of IA32_PERF_CAPABILITIES are defined in Table Table 2-2. "IA-32 Architectural MSRs" of Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4. The relevant bit fields that governs PEBS are:

- **PEBSTrap [bit 6]:** When set, PEBS recording is trap-like. After the PEBS-enabled counter has overflowed, PEBS record is recorded for the next PEBS-able event at the completion of the sampled instruction causing the PEBS event. When clear, PEBS recording is fault-like. The PEBS record is recorded before the sampled instruction causing the PEBS event.
- **PEBSSaveArchRegs [bit 7]:** When set, PEBS will save architectural register and state information according to the encoded value of the PEBSRecordFormat field. When clear, only the return instruction pointer and flags are recorded. On processors based on Intel Core microarchitecture, this bit is always 1.
- **PEBSRecordFormat [bits 11:8]:** Valid encodings are:
 - 0000B: Only general-purpose registers, instruction pointer and RFLAGS registers are saved in each PEBS record (See Section 22.6.3.8).
 - 0001B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS and load latency data. (See Section 22.3.1.1.1).
 - 0010B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS, load latency data, and TSX tuning information. (See Section 22.3.6.2).
 - 0011B: PEBS record includes additional information of load latency data, TSX tuning information, TSC data, and the applicable counter field replaces IA32_PERF_GLOBAL_STATUS at offset 90H. (See Section 22.3.8.1.1).

- 0100B: PEBS record contents are defined by elections in MSR_PEBS_DATA_CFG. (See Section 22.9.2.3). The PEBS Configuration Buffer is defined as shown in Figure 22-66 with Counter Reset fields allocation for 8 general-purpose counters followed by 4 fixed-function counters.
- 0101B: PEBS record contents are defined by elections in MSR_PEBS_DATA_CFG. (See Section 22.9.2.3). The PEBS Configuration Buffer is defined as shown in Figure 22-66 with Counter Reset fields allocation for 32 general-purpose counters followed by 16 fixed-function counters.
- 0110B: PEBS record contents are defined by elections in MSR_PEBS_DATA_CFG (see Figure 22-73 in Section 22.9.2.3) that is compatible with the previous MSR_PEBS_DATA_CFG (see Figure 22-72 in Section 22.9.2.3). The PEBS Config Buffer is defined as shown in Figure 22-72 with a Counter Reset fields allocation for 32 general-purpose counters followed by 16 fixed-function counters.

22.6.2.4.3 Writing a PEBS Interrupt Service Routine

The PEBS facilities share the same interrupt vector and interrupt service routine (called the DS ISR) with the Interrupt-based event sampling and BTS facilities. To handle PEBS interrupts, PEBS handler code must be included in the DS ISR. See Section 20.4.9.1, "64 Bit Format of the DS Save Area," for guidelines when writing the DS ISR.

The service routine can query MSR_PERF_GLOBAL_STATUS to determine which counter(s) caused of overflow condition. The service routine should clear overflow indicator by writing to MSR_PERF_GLOBAL_OVF_CTL.

A comparison of the sequence of requirements to program PEBS for processors based on Intel Core and Intel NetBurst microarchitectures is listed in Table 22-89. "Requirements to Program PEBS".

Table 22-89. Requirements to Program PEBS

	For Processors based on Intel Core microarchitecture	For Processors based on Intel NetBurst microarchitecture
Verify PEBS support of processor/OS.	▪ IA32_MISC_ENABLE.EMON_AVAILABE (bit 7) is set. ▪ IA32_MISC_ENABLE.PEBS_UNAVAILABE (bit 12) is clear.	
Ensure counters are in disabled.	On initial set up or changing event configurations, write MSR_PERF_GLOBAL_CTRL MSR (38FH) with 0. On subsequent entries: ▪ Clear all counters if "Counter Freeze on PMI" is not enabled. ▪ If IA32_DebugCTL.Freeze is enabled, counters are automatically disabled. Counters MUST be stopped before writing. ¹	Optional
Disable PEBS.	Clear ENABLE PMCO bit in IA32_PEBS_ENABLE MSR (3F1H).	Optional
Check overflow conditions.	Check MSR_PERF_GLOBAL_STATUS MSR (38EH) handle any overflow conditions.	Check OVF flag of each CCCR for overflow condition
Clear overflow status.	Clear MSR_PERF_GLOBAL_STATUS MSR (38EH) using IA32_PERF_GLOBAL_OVF_CTRL MSR (390H).	Clear OVF flag of each CCCR.
Write "sample-after" values.	Configure the counter(s) with the sample after value.	
Configure specific counter configuration MSR.	▪ Set local enable bit 22 - 1. ▪ Do NOT set local counter PMI/INT bit, bit 20 - 0. ▪ Event programmed must be PEBS capable.	▪ Set appropriate OVF_PMI bits - 1. ▪ Only CCCR for MSR_IQ_COUNTER4 support PEBS.
Allocate buffer for PEBS states.	Allocate a buffer in memory for the precise information.	
Program the IA32_DS_AREA MSR.	Program the IA32_DS_AREA MSR.	
Configure the PEBS buffer management records.	Configure the PEBS buffer management records in the DS buffer management area.	
Configure/Enable PEBS.	Set Enable PMCO bit in IA32_PEBS_ENABLE MSR (3F1H).	Configure MSR_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT, and MSR_PEBS_MATRIX_HORZ as needed.

Table 22-89. Requirements to Program PEBS (Contd.)

	For Processors based on Intel Core microarchitecture	For Processors based on Intel NetBurst microarchitecture
Enable counters.	Set Enable bits in MSR_PERF_GLOBAL_CTRL MSR (38FH).	Set each CCCR enable bit 12 - 1.

NOTES:

1. Counters read while enabled are not guaranteed to be precise with event counts that occur in timing proximity to the RDMSR.

22.6.2.4.4 Re-configuring PEBS Facilities

When software needs to reconfigure PEBS facilities, it should allow a quiescent period between stopping the prior event counting and setting up a new PEBS event. The quiescent period is to allow any latent residual PEBS records to complete its capture at their previously specified buffer address (provided by IA32_DS_AREA).

22.6.3 Performance Monitoring (Processors Based on Intel NetBurst® Microarchitecture)

The performance monitoring mechanism provided in processors based on Intel NetBurst microarchitecture is different from that provided in the P6 family and Pentium processors. While the general concept of selecting, filtering, counting, and reading performance events through the WRMSR, RDMSR, and RDPMC instructions is unchanged, the setup mechanism and MSR layouts are incompatible with the P6 family and Pentium processor mechanisms. Also, the RDPMC instruction has been extended to support faster reading of counters and to read all performance counters available in processors based on Intel NetBurst microarchitecture.

The event monitoring mechanism consists of the following facilities:

- The IA32_MISC_ENABLE MSR, which indicates the availability in an Intel 64 or IA-32 processor of the performance monitoring and processor event-based sampling (PEBS) facilities.
- Event selection control (ESCR) MSRs for selecting events to be monitored with specific performance counters. The number available differs by family and model (43 to 45).
- 18 performance counter MSRs for counting events.
- 18 counter configuration control (CCCR) MSRs, with one CCCR associated with each performance counter. CCCR sets up an associated performance counter for a specific method of counting.
- A debug store (DS) save area in memory for storing PEBS records.
- The IA32_DS_AREA MSR, which establishes the location of the DS save area.
- The debug store (DS) feature flag (bit 21) returned by the CPUID instruction, which indicates the availability of the DS mechanism.
- The MSR_PEBS_ENABLE MSR, which enables the PEBS facilities and replay tagging used in at-retirement event counting.
- A set of predefined events and event metrics that simplify the setting up of the performance counters to count specific events.

Table 22-90 lists the performance counters and their associated CCCRs, along with the ESCRs that select events to be counted for each performance counter. Predefined event metrics and events can be found at: <https://perfmon-events.intel.com/>.

**Table 22-90. Performance Counter MSRs and Associated CCCR and ESCR MSRs
(Processors Based on Intel NetBurst Microarchitecture)**

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_BPU_COUNTER0	0	300H	MSR_BPU_CCCR0	360H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H
MSR_BPU_COUNTER1	1	301H	MSR_BPU_CCCR1	361H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H
MSR_BPU_COUNTER2	2	302H	MSR_BPU_CCCR2	362H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_BPU_COUNTER3	3	303H	MSR_BPU_CCCR3	363H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_MS_COUNTER0	4	304H	MSR_MS_CCCR0	364H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER1	5	305H	MSR_MS_CCCR1	365H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER2	6	306H	MSR_MS_CCCR2	366H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_MS_COUNTER3	7	307H	MSR_MS_CCCR3	367H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_FLAME_COUNTER0	8	308H	MSR_FLAME_CCCR0	368H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAA_T_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_COUNTER1	9	309H	MSR_FLAME_CCCR1	369H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAA_T_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H

**Table 22-90. Performance Counter MSRs and Associated CCCR and ESCR MSRs
(Processors Based on Intel NetBurst Microarchitecture) (Contd.)**

Counter			CCCR		ESCR		
Name	No.	Addr	Name	Addr	Name	No.	Addr
MSR_FLAME_COUNTER2	10	30AH	MSR_FLAME_CCCR2	36AH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAAT_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_FLAME_COUNTER3	11	30BH	MSR_FLAME_CCCR3	36BH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAAT_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_IQ_COUNTER0	12	30CH	MSR_IQ_CCCR0	36CH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0 ¹ MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER1	13	30DH	MSR_IQ_CCCR1	36DH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0 ¹ MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER2	14	30EH	MSR_IQ_CCCR2	36EH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 ¹ MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER3	15	30FH	MSR_IQ_CCCR3	36FH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 ¹ MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER4	16	310H	MSR_IQ_CCCR4	370H	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0 ¹ MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER5	17	311H	MSR_IQ_CCCR5	371H	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1 ¹ MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH

NOTES:

1. MSR_IQ_ESCR0 and MSR_IQ_ESCR1 are available only on early processor builds (family 0FH, models 01H-02H). These MSRs are not available on later versions.

The types of events that can be counted with these performance monitoring facilities are divided into two classes: non-retirement events and at-retirement events.

- Non-retirement events are events that occur any time during instruction execution (such as bus transactions or cache transactions).
- At-retirement events are events that are counted at the retirement stage of instruction execution, which allows finer granularity in counting events and capturing machine state.

The at-retirement counting mechanism includes facilities for tagging μ ops that have encountered a particular performance event during instruction execution. Tagging allows events to be sorted between those that occurred on an execution path that resulted in architectural state being committed at retirement as well as events that occurred on an execution path where the results were eventually cancelled and never committed to architectural state (such as, the execution of a mispredicted branch).

The Pentium 4 and Intel Xeon processor performance monitoring facilities support the three usage models described below. The first two models can be used to count both non-retirement and at-retirement events; the third model is used to count a subset of at-retirement events:

- **Event counting** — A performance counter is configured to count one or more types of events. While the counter is counting, software reads the counter at selected intervals to determine the number of events that have been counted between the intervals.
- **Interrupt-based event sampling** — A performance counter is configured to count one or more types of events and to generate an interrupt when it overflows. To trigger an overflow, the counter is preset to a modulus value that will cause the counter to overflow after a specific number of events have been counted.
When the counter overflows, the processor generates a performance monitoring interrupt (PMI). The interrupt service routine for the PMI then records the return instruction pointer (RIP), resets the modulus, and restarts the counter. Code performance can be analyzed by examining the distribution of RIPs with a tool like the VTune™ Performance Analyzer.
- **Processor event-based sampling (PEBS)** — In PEBS, the processor writes a record of the architectural state of the processor to a memory buffer after the counter overflows. The records of architectural state provide additional information for use in performance tuning. Processor-based event sampling can be used to count only a subset of at-retirement events. PEBS captures more precise processor state information compared to interrupt based event sampling, because the latter need to use the interrupt service routine to re-construct the architectural states of processor.

The following sections describe the MSRs and data structures used for performance monitoring in the Pentium 4 and Intel Xeon processors.

22.6.3.1 ESCR MSRs

The 45 ESCR MSRs (see Table 22-90) allow software to select specific events to be countered. Each ESCR is usually associated with a pair of performance counters (see Table 22-90) and each performance counter has several ESCRs associated with it (allowing the events counted to be selected from a variety of events).

Figure 22-49 shows the layout of an ESCR MSR. The functions of the flags and fields are:

- **USR flag, bit 2** — When set, events are counted when the processor is operating at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.
- **OS flag, bit 3** — When set, events are counted when the processor is operating at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the OS and USR flags are set, events are counted at all privilege levels.)

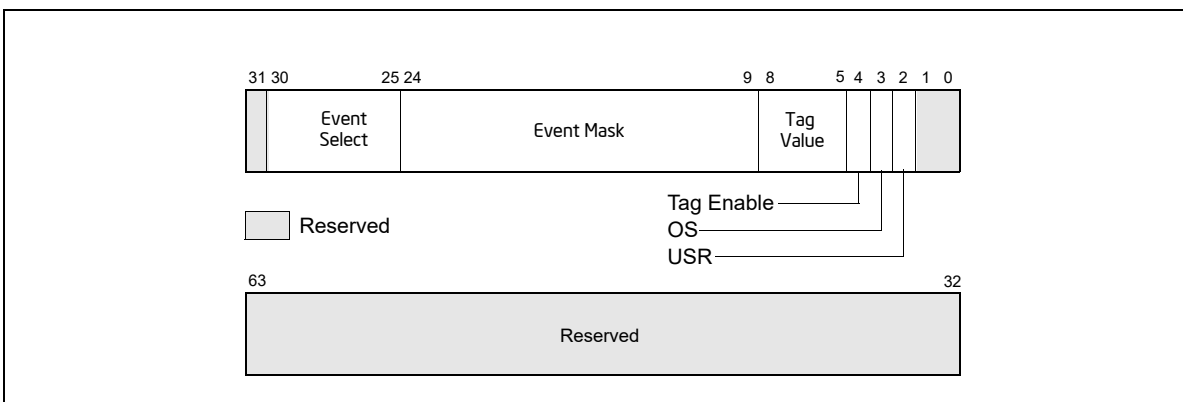


Figure 22-49. Event Selection Control Register (ESCR) for Pentium 4 and Intel® Xeon® Processors without Intel HT Technology Support

- **Tag enable, bit 4** — When set, enables tagging of μ ops to assist in at-retirement event counting; when clear, disables tagging. See Section 22.6.3.6, “At-Retirement Counting.”
- **Tag value field, bits 5 through 8** — Selects a tag value to associate with a μ op to assist in at-retirement event counting.
- **Event mask field, bits 9 through 24** — Selects events to be counted from the event class selected with the event select field.
- **Event select field, bits 25 through 30** — Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

When setting up an ESCR, the event select field is used to select a specific class of events to count, such as retired branches. The event mask field is then used to select one or more of the specific events within the class to be counted. For example, when counting retired branches, four different events can be counted: branch not taken predicted, branch not taken mispredicted, branch taken predicted, and branch taken mispredicted. The OS and USR flags allow counts to be enabled for events that occur when operating system code and/or application code are being executed. If neither the OS nor USR flag is set, no events will be counted.

The ESCRs are initialized to all 0s on reset. The flags and fields of an ESCR are configured by writing to the ESCR using the WRMSR instruction. Table 22-90 gives the addresses of the ESCR MSRs.

Writing to an ESCR MSR does not enable counting with its associated performance counter; it only selects the event or events to be counted. The CCCR for the selected performance counter must also be configured. Configuration of the CCCR includes selecting the ESCR and enabling the counter.

22.6.3.2 Performance Counters

The performance counters in conjunction with the counter configuration control registers (CCCRs) are used for filtering and counting the events selected by the ESCRs. Processors based on Intel NetBurst microarchitecture provide 18 performance counters organized into 9 pairs. A pair of performance counters is associated with a particular subset of events and ESCR's (see Table 22-90). The counter pairs are partitioned into four groups:

- The BPU group, includes two performance counter pairs:
 - MSR_BPU_COUNTER0 and MSR_BPU_COUNTER1.
 - MSR_BPU_COUNTER2 and MSR_BPU_COUNTER3.
- The MS group, includes two performance counter pairs:
 - MSR_MS_COUNTER0 and MSR_MS_COUNTER1.
 - MSR_MS_COUNTER2 and MSR_MS_COUNTER3.
- The FLAME group, includes two performance counter pairs:
 - MSR_FLAME_COUNTER0 and MSR_FLAME_COUNTER1.

- MSR_FLAME_COUNTER2 and MSR_FLAME_COUNTER3.
- The IQ group, includes three performance counter pairs:
 - MSR_IQ_COUNTER0 and MSR_IQ_COUNTER1.
 - MSR_IQ_COUNTER2 and MSR_IQ_COUNTER3.
 - MSR_IQ_COUNTER4 and MSR_IQ_COUNTER5.

The MSR_IQ_COUNTER4 counter in the IQ group provides support for the PEBS.

Alternate counters in each group can be cascaded: the first counter in one pair can start the first counter in the second pair and vice versa. A similar cascading is possible for the second counters in each pair. For example, within the BPU group of counters, MSR_BPU_COUNTER0 can start MSR_BPU_COUNTER2 and vice versa, and MSR_BPU_COUNTER1 can start MSR_BPU_COUNTER3 and vice versa (see Section 22.6.3.5.6, “Cascading Counters”). The cascade flag in the CCCR register for the performance counter enables the cascading of counters.

Each performance counter is 40-bits wide (see Figure 22-50). The RDPMC instruction is intended to allow reading of either the full counter-width (40-bits) or, if ECX[31] is set to 1, the low 32-bits of the counter. Reading the low 32-bits is faster than reading the full counter width and is appropriate in situations where the count is small enough to be contained in 32 bits. In such cases, counter bits 31:0 are written to EAX, while 0 is written to EDX.

The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

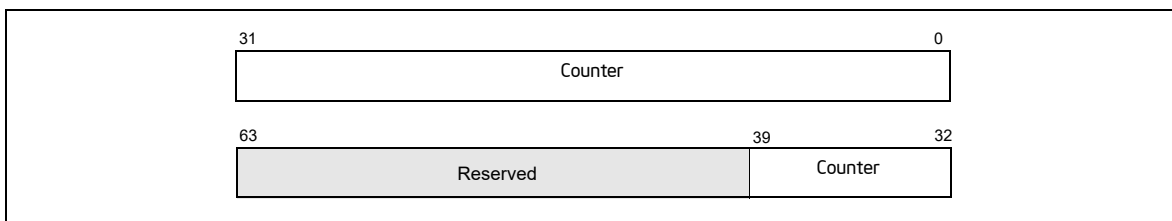


Figure 22-50. Performance Counter (Pentium 4 and Intel® Xeon® Processors)

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

Some uses of the performance counters require the counters to be preset before counting begins (that is, before the counter is enabled). This can be accomplished by writing to the counter using the WRMSR instruction. To set a counter to a specified number of counts before overflow, enter a 2s complement negative integer in the counter. The counter will then count from the preset value up to -1 and overflow. Writing to a performance counter in a Pentium 4 or Intel Xeon processor with the WRMSR instruction causes all 40 bits of the counter to be written.

22.6.3.3 CCCR MSRs

Each of the 18 performance counters has one CCCR MSR associated with it (see Table 22-90). The CCCRs control the filtering and counting of events as well as interrupt generation. Figure 22-51 shows the layout of an CCCR MSR. The functions of the flags and fields are as follows:

- **Enable flag, bit 12** — When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset.
- **ESCR select field, bits 13 through 15** — Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.

- **Compare flag, bit 18** — When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.
- **Complement flag, bit 19** — Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance counter; when clear, counts greater than the threshold value result in a count being delivered to the performance counter (see Section 22.6.3.5.2, “Filtering Events”). The complement flag is not active unless the compare flag is set.
- **Threshold field, bits 20 through 23** — Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 22.6.3.5.2, “Filtering Events”).
- **Edge flag, bit 24** — When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.

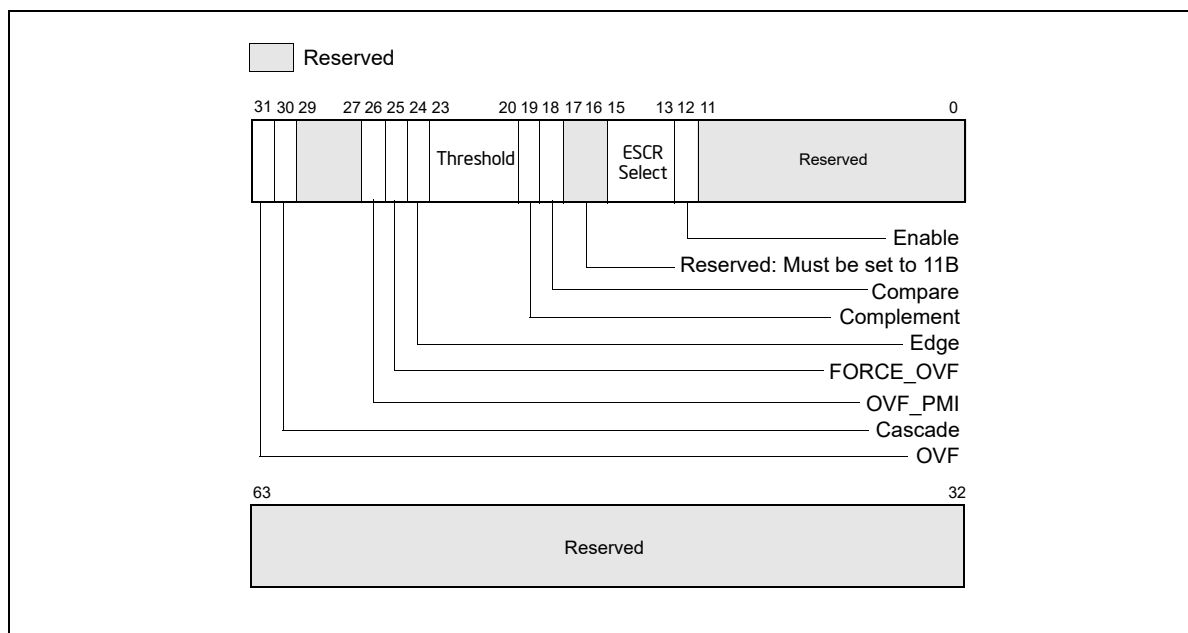


Figure 22-51. Counter Configuration Control Register (CCCR)

- **FORCE_OVF flag, bit 25** — When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.
- **OVF_PMI flag, bit 26** — When set, causes a performance monitor interrupt (PMI) to be generated when the counter overflows occurs; when clear, disables PMI generation. Note that the PMI is generated on the next event count after the counter has overflowed.
- **Cascade flag, bit 30** — When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 22.6.3.2, “Performance Counters,” for further details); when clear, disables cascading of counters.
- **OVF flag, bit 31** — Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

The CCCRs are initialized to all 0s on reset.

The events that an enabled performance counter actually counts are selected and filtered by the following flags and fields in the ESCR and CCCR registers and in the qualification order given:

1. The event select and event mask fields in the ESCR select a class of events to be counted and one or more event types within the class, respectively.

2. The OS and USR flags in the ESCR selected the privilege levels at which events will be counted.
3. The ESCR select field of the CCCR selects the ESCR. Since each counter has several ESCRs associated with it, one ESCR must be chosen to select the classes of events that may be counted.
4. The compare and complement flags and the threshold field of the CCCR select an optional threshold to be used in qualifying an event count.
5. The edge flag in the CCCR allows events to be counted only on rising-edge transitions.

The qualification order in the above list implies that the filtered output of one “stage” forms the input for the next. For instance, events filtered using the privilege level flags can be further qualified by the compare and complement flags and the threshold field, and an event that matched the threshold criteria, can be further qualified by edge detection.

The uses of the flags and fields in the CCCRs are discussed in greater detail in Section 22.6.3.5, “Programming the Performance Counters for Non-Retirement Events.”

22.6.3.4 Debug Store (DS) Mechanism

The debug store (DS) mechanism was introduced with processors based on Intel NetBurst microarchitecture to allow various types of information to be collected in memory-resident buffers for use in debugging and tuning programs. The DS mechanism can be used to collect two types of information: branch records and processor event-based sampling (PEBS) records. The availability of the DS mechanism in a processor is indicated with the DS feature flag (bit 21) returned by the CPUID instruction.

See Section 20.4.5, “Branch Trace Store (BTS),” and Section 22.6.3.8, “Processor Event-Based Sampling (PEBS),” for a description of these facilities. Records collected with the DS mechanism are saved in the DS save area. See Section 20.4.9, “BTS and DS Save Area.”

22.6.3.5 Programming the Performance Counters for Non-Retirement Events

The basic steps to program a performance counter and to count events include the following:

1. Select the event or events to be counted.
2. For each event, select an ESCR that supports the event.
3. Match the CCCR Select value and ESCR name to a value listed in Table 22-90; select a CCCR and performance counter.
4. Set up an ESCR for the specific event or events to be counted and the privilege levels at which they are to be counted.
5. Set up the CCCR for the performance counter by selecting the ESCR and the desired event filters.
6. Set up the CCCR for optional cascading of event counts, so that when the selected counter overflows its alternate counter starts.
7. Set up the CCCR to generate an optional performance monitor interrupt (PMI) when the counter overflows. If PMI generation is enabled, the local APIC must be set up to deliver the interrupt to the processor and a handler for the interrupt must be in place.
8. Enable the counter to begin counting.

22.6.3.5.1 Selecting Events to Count

There is a set of at-retirement events for processors based on Intel NetBurst microarchitecture. For each event, setup information is provided. Table 22-91 gives an example of one of the events.

Table 22-91. Event Example

Event Name	Event Parameters	Parameter Value	Description
branch_retired			Counts the retirement of a branch. Specify one or more mask bits to select any combination of branch taken, not-taken, predicted, and mispredicted.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	See Table 15-3 for the addresses of the ESCR MSRs.
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	The counter numbers associated with each ESCR are provided. The performance counters and corresponding CCCRs can be obtained from Table 15-3.
	ESCR Event Select	06H	ESCR[31:25]
	ESCR Event Mask	Bit 0: MMNP 1: MMNM 2: MMTP 3: MMTM	ESCR[24:9] Branch Not-taken Predicted Branch Not-taken Mispredicted Branch Taken Predicted Branch Taken Mispredicted
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		P6: EMON_BR_INST_RETIRED
	Can Support PEBS	No	
	Requires Additional MSRs for Tagging	No	

Event Parameters are described below.

- **ESCR restrictions** — Lists the ESCRs that can be used to program the event. Typically only one ESCR is needed to count an event.
- **Counter numbers per ESCR** — Lists which performance counters are associated with each ESCR. Table 22-90 gives the name of the counter and CCCR for each counter number. Typically only one counter is needed to count the event.
- **ESCR event select** — Gives the value to be placed in the event select field of the ESCR to select the event.
- **ESCR event mask** — Gives the value to be placed in the Event Mask field of the ESCR to select sub-events to be counted. The parameter value column defines the documented bits with relative bit position offset starting from 0, where the absolute bit position of relative offset 0 is bit 9 of the ESCR. All undocumented bits are reserved and should be set to 0.
- **CCCR select** — Gives the value to be placed in the ESCR select field of the CCCR associated with the counter to select the ESCR to be used to define the event. This value is not the address of the ESCR; it is the number of the ESCR from the Number column in Table 22-90.
- **Event specific notes** — Gives additional information about the event, such as the name of the same or a similar event defined for the P6 family processors.
- **Can support PEBS** — Indicates if PEBS is supported for the event (only supplied for at-retirement events).
- **Requires additional MSR for tagging** — Indicates which if any additional MSRs must be programmed to count the events (only supplied for the at-retirement events).

NOTE

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. The counter values reported are not guaranteed to be

absolutely accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The following procedure shows how to set up a performance counter for basic counting; that is, the counter is set up to count a specified event indefinitely, wrapping around whenever it reaches its maximum count. This procedure is continued through the following four sections.

An event to be counted can be selected as follows:

1. Select the event to be counted.
2. Select the ESCR to be used to select events to be counted from the ESCRs field.
3. Select the number of the counter to be used to count the event from the Counter Numbers Per ESCR field.
4. Determine the name of the counter and the CCCR associated with the counter, and determine the MSR addresses of the counter, CCCR, and ESCR from Table 22-90.
5. Use the WRMSR instruction to write the ESCR Event Select and ESCR Event Mask values into the appropriate fields in the ESCR. At the same time set or clear the USR and OS flags in the ESCR as desired.
6. Use the WRMSR instruction to write the CCCR Select value into the appropriate field in the CCCR.

NOTE

Typically all the fields and flags of the CCCR will be written with one WRMSR instruction; however, in this procedure, several WRMSR writes are used to more clearly demonstrate the uses of the various CCCR fields and flags.

This setup procedure is continued in the next section, Section 22.6.3.5.2, "Filtering Events."

22.6.3.5.2 Filtering Events

Each counter receives up to 4 input lines from the processor hardware from which it is counting events. The counter treats these inputs as binary inputs (input 0 has a value of 1, input 1 has a value of 2, input 2 has a value of 4, and input 3 has a value of 8). When a counter is enabled, it adds this binary input value to the counter value on each clock cycle. For each clock cycle, the value added to the counter can then range from 0 (no event) to 15.

For many events, only the 0 input line is active, so the counter is merely counting the clock cycles during which the 0 input is asserted. However, for some events two or more input lines are used. Here, the counter's threshold setting can be used to filter events. The compare, complement, threshold, and edge fields control the filtering of counter increments by input value.

If the compare flag is set, then a "greater than" or a "less than or equal to" comparison of the input value vs. a threshold value can be made. The complement flag selects "less than or equal to" (flag set) or "greater than" (flag clear). The threshold field selects a threshold value of from 0 to 15. For example, if the complement flag is cleared and the threshold field is set to 6, then any input value of 7 or greater on the 4 inputs to the counter will cause the counter to be incremented by 1, and any value less than 7 will cause an increment of 0 (or no increment) of the counter. Conversely, if the complement flag is set, any value from 0 to 6 will increment the counter and any value from 7 to 15 will not increment the counter. Note that when a threshold condition has been satisfied, the input to the counter is always 1, not the input value that is presented to the threshold filter.

The edge flag provides further filtering of the counter inputs when a threshold comparison is being made. The edge flag is only active when the compare flag is set. When the edge flag is set, the resulting output from the threshold filter (a value of 0 or 1) is used as an input to the edge filter. Each clock cycle, the edge filter examines the last and current input values and sends a count to the counter only when it detects a "rising edge" event; that is, a false-to-true transition. Figure 22-52 illustrates rising edge filtering.

The following procedure shows how to configure a CCCR to filter events using the threshold filter and the edge filter. This procedure is a continuation of the setup procedure introduced in Section 22.6.3.5.1, "Selecting Events to Count."

7. (Optional) To set up the counter for threshold filtering, use the WRMSR instruction to write values in the CCCR compare and complement flags and the threshold field:

- Set the compare flag.
- Set or clear the complement flag for less than or equal to or greater than comparisons, respectively.
- Enter a value from 0 to 15 in the threshold field.

8. (Optional) Select rising edge filtering by setting the CCCR edge flag.

This setup procedure is continued in the next section, Section 22.6.3.5.3, “Starting Event Counting.”

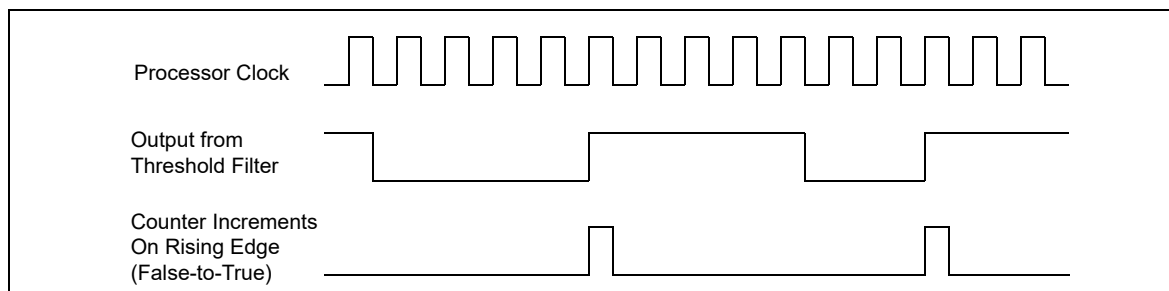


Figure 22-52. Effects of Edge Filtering

22.6.3.5.3 Starting Event Counting

Event counting by a performance counter can be initiated in either of two ways. The typical way is to set the enable flag in the counter’s CCCR. Following the instruction to set the enable flag, event counting begins and continues until it is stopped (see Section 22.6.3.5.5, “Halting Event Counting”).

The following procedural step shows how to start event counting. This step is a continuation of the setup procedure introduced in Section 22.6.3.5.2, “Filtering Events.”

9. To start event counting, use the WRMSR instruction to set the CCCR enable flag for the performance counter.

This setup procedure is continued in the next section, Section 22.6.3.5.4, “Reading a Performance Counter’s Count.”

The second way that a counter can be started by using the cascade feature. Here, the overflow of one counter automatically starts its alternate counter (see Section 22.6.3.5.6, “Cascading Counters”).

22.6.3.5.4 Reading a Performance Counter’s Count

Performance counters can be read using either the RDPMC or RDMSR instructions. The enhanced functions of the RDPMC instruction (including fast read) are described in Section 22.6.3.2, “Performance Counters.” These instructions can be used to read a performance counter while it is counting or when it is stopped.

The following procedural step shows how to read the event counter. This step is a continuation of the setup procedure introduced in Section 22.6.3.5.3, “Starting Event Counting.”

10. To read a performance counters current event count, execute the RDPMC instruction with the counter number obtained from Table 22-90 used as an operand.

This setup procedure is continued in the next section, Section 22.6.3.5.5, “Halting Event Counting.”

22.6.3.5.5 Halting Event Counting

After a performance counter has been started (enabled), it continues counting indefinitely. If the counter overflows (goes one count past its maximum count), it wraps around and continues counting. When the counter wraps around, it sets its OVF flag to indicate that the counter has overflowed. The OVF flag is a sticky flag that indicates that the counter has overflowed at least once since the OVF bit was last cleared.

To halt counting, the CCCR enable flag for the counter must be cleared.

The following procedural step shows how to stop event counting. This step is a continuation of the setup procedure introduced in Section 22.6.3.5.4, “Reading a Performance Counter’s Count.”

11. To stop event counting, execute a WRMSR instruction to clear the CCCR enable flag for the performance counter.

To halt a cascaded counter (a counter that was started when its alternate counter overflowed), either clear the Cascade flag in the cascaded counter's CCCR MSR or clear the OVF flag in the alternate counter's CCCR MSR.

22.6.3.5.6 Cascading Counters

As described in Section 22.6.3.2, "Performance Counters," eighteen performance counters are implemented in pairs. Nine pairs of counters and associated CCCRs are further organized as four blocks: BPU, MS, FLAME, and IQ (see Table 22-90). The first three blocks contain two pairs each. The IQ block contains three pairs of counters (12 through 17) with associated CCCRs (MSR_IQ_CCCR0 through MSR_IQ_CCCR5).

The first 8 counter pairs (0 through 15) can be programmed using ESCRs to detect performance monitoring events. Pairs of ESCRs in each of the four blocks allow many different types of events to be counted. The cascade flag in the CCCR MSR allows nested monitoring of events to be performed by cascading one counter to a second counter located in another pair in the same block (see Figure 22-51 for the location of the flag).

Counters 0 and 1 form the first pair in the BPU block. Either counter 0 or 1 can be programmed to detect an event via MSR_MO B_ESCR0. Counters 0 and 2 can be cascaded in any order, as can counters 1 and 3. It's possible to set up 4 counters in the same block to cascade on two pairs of independent events. The pairing described also applies to subsequent blocks. Since the IQ PUB has two extra counters, cascading operates somewhat differently if 16 and 17 are involved. In the IQ block, counter 16 can only be cascaded from counter 14 (not from 12); counter 14 cannot be cascaded from counter 16 using the CCCR cascade bit mechanism. Similar restrictions apply to counter 17.

Example 22-1. Counting Events

Assume a scenario where counter X is set up to count 200 occurrences of event A; then counter Y is set up to count 400 occurrences of event B. Each counter is set up to count a specific event and overflow to the next counter. In the above example, counter X is preset for a count of -200 and counter Y for a count of -400; this setup causes the counters to overflow on the 200th and 400th counts respectively.

Continuing this scenario, counter X is set up to count indefinitely and wraparound on overflow. This is described in the basic performance counter setup procedure that begins in Section 22.6.3.5.1, "Selecting Events to Count." Counter Y is set up with the cascade flag in its associated CCCR MSR set to 1 and its enable flag set to 0.

To begin the nested counting, the enable bit for the counter X is set. Once enabled, counter X counts until it overflows. At this point, counter Y is automatically enabled and begins counting. Thus counter X overflows after 200 occurrences of event A. Counter Y then starts, counting 400 occurrences of event B before overflowing. When performance counters are cascaded, the counter Y would typically be set up to generate an interrupt on overflow. This is described in Section 22.6.3.5.8, "Generating an Interrupt on Overflow."

The cascading counters mechanism can be used to count a single event. The counting begins on one counter then continues on the second counter after the first counter overflows. This technique doubles the number of event counts that can be recorded, since the contents of the two counters can be added together.

22.6.3.5.7 EXTENDED CASCADING

Extended cascading is a model-specific feature in the Intel NetBurst microarchitecture with CPUID DisplayFamily_DisplayModel 0F_02, 0F_03, 0F_04, 0F_06. This feature uses bit 11 in CCCRs associated with the IQ block. See Table 22-92.

Table 22-92. CCR Names and Bit Positions

CCCR Name:Bit Position	Bit Name	Description
MSR_IQ_CCCR1 2:11	Reserved	
MSR_IQ_CCCR0:11	CASCNT4INT00	Allow counter 4 to cascade into counter 0
MSR_IQ_CCCR3:11	CASCNT5INT03	Allow counter 5 to cascade into counter 3

Table 22-92. CCR Names and Bit Positions (Contd.)

MSR_IQ_CCCR4:11	CASCNT5INTO4	Allow counter 5 to cascade into counter 4
MSR_IQ_CCCR5:11	CASCNT4INTO5	Allow counter 4 to cascade into counter 5

The extended cascading feature can be adapted to the Interrupt based sampling usage model for performance monitoring. However, it is known that performance counters do not generate PMI in cascade mode or extended cascade mode due to an erratum. This erratum applies to processors with CPUID DisplayFamily_DisplayModel signature of 0F_02. For processors with CPUID DisplayFamily_DisplayModel signature of 0F_00 and 0F_01, the erratum applies to processors with stepping encoding greater than 09H.

Counters 16 and 17 in the IQ block are frequently used in processor event-based sampling or at-retirement counting of events indicating a stalled condition in the pipeline. Neither counter 16 or 17 can initiate the cascading of counter pairs using the cascade bit in a CCCR.

Extended cascading permits performance monitoring tools to use counters 16 and 17 to initiate cascading of two counters in the IQ block. Extended cascading from counter 16 and 17 is conceptually similar to cascading other counters, but instead of using CASCADE bit of a CCCR, one of the four CASCNTxINTOy bits is used.

Example 22-2. Scenario for Extended Cascading

A usage scenario for extended cascading is to sample instructions retired on logical processor 1 after the first 4096 instructions retired on logical processor 0. A procedure to program extended cascading in this scenario is outlined below:

1. Write the value 0 to counter 12.
2. Write the value 04000603H to MSR_CRU_ESCR0 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 1).
3. Write the value 04038800H to MSR_IQ_CCCR0. This enables CASCNT4INTO0 and OVF_PMI. An ISR can sample on instruction addresses in this case (do not set ENABLE, or CASCADE).
4. Write the value FFFF000H into counter 16.1.
5. Write the value 0400060CH to MSR_CRU_ESCR2 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 0).
6. Write the value 00039000H to MSR_IQ_CCCR4 (set ENABLE bit, but not OVF_PMI).

Another use for cascading is to locate stalled execution in a multithreaded application. Assume MOB replays in thread B cause thread A to stall. Getting a sample of the stalled execution in this scenario could be accomplished by:

1. Set up counter B to count MOB replays on thread B.
2. Set up counter A to count resource stalls on thread A; set its force overflow bit and the appropriate CASCNTx-INTOy bit.
3. Use the performance monitoring interrupt to capture the program execution data of the stalled thread.

22.6.3.5.8 Generating an Interrupt on Overflow

Any performance counter can be configured to generate a performance monitor interrupt (PMI) if the counter overflows. The PMI interrupt service routine can then collect information about the state of the processor or program when overflow occurred. This information can then be used with a tool like the Intel® VTune™ Performance Analyzer to analyze and tune program performance.

To enable an interrupt on counter overflow, the OVR_PMI flag in the counter's associated CCCR MSR must be set. When overflow occurs, a PMI is generated through the local APIC. (Here, the performance counter entry in the local vector table [LVT] is set up to deliver the interrupt generated by the PMI to the processor.)

The PMI service routine can use the OVF flag to determine which counter overflowed when multiple counters have been configured to generate PMIs. Also, note that these processors mask PMIs upon receiving an interrupt. Clear this condition before leaving the interrupt handler.

When generating interrupts on overflow, the performance counter being used should be preset to value that will cause an overflow after a specified number of events are counted plus 1. The simplest way to select the preset value is to write a negative number into the counter, as described in Section 22.6.3.5.6, “Cascading Counters.” Here, however, if an interrupt is to be generated after 100 event counts, the counter should be preset to minus 100 plus 1 ($-100 + 1$), or -99. The counter will then overflow after it counts 99 events and generate an interrupt on the next (100th) event counted. The difference of 1 for this count enables the interrupt to be generated immediately after the selected event count has been reached, instead of waiting for the overflow to be propagation through the counter.

Because of latency in the microarchitecture between the generation of events and the generation of interrupts on overflow, it is sometimes difficult to generate an interrupt close to an event that caused it. In these situations, the `FORCE_OVF` flag in the CCCR can be used to improve reporting. Setting this flag causes the counter to overflow on every counter increment, which in turn triggers an interrupt after every counter increment.

22.6.3.5.9 Counter Usage Guideline

There are some instances where the user must take care to configure counting logic properly, so that it is not powered down. To use any ESCR, even when it is being used just for tagging, (any) one of the counters that the particular ESCR (or its paired ESCR) can be connected to should be enabled. If this is not done, 0 counts may result. Likewise, to use any counter, there must be some event selected in a corresponding ESCR (other than `no_event`, which generally has a select value of 0).

22.6.3.6 At-Retirement Counting

At-retirement counting provides a means counting only events that represent work committed to architectural state and ignoring work that was performed speculatively and later discarded.

One example of this speculative activity is branch prediction. When a branch misprediction occurs, the results of instructions that were decoded and executed down the mispredicted path are canceled. If a performance counter was set up to count all executed instructions, the count would include instructions whose results were canceled as well as those whose results committed to architectural state.

To provide finer granularity in event counting in these situations, the performance monitoring facilities provided in the Pentium 4 and Intel Xeon processors provide a mechanism for tagging events and then counting only those tagged events that represent committed results. This mechanism is called “at-retirement counting.”

There are predefined at-retirement events and event metrics that can be used to for tagging events when using at retirement counting. The following terminology is used in describing at-retirement counting:

- **Bogus, non-bogus, retire** — In at-retirement event descriptions, the term “bogus” refers to instructions or μ ops that must be canceled because they are on a path taken from a mispredicted branch. The terms “retired” and “non-bogus” refer to instructions or μ ops along the path that results in committed architectural state changes as required by the program being executed. Thus instructions and μ ops are either bogus or non-bogus, but not both. Several of the Pentium 4 and Intel Xeon processors’ performance monitoring events (such as, `Instruction_Retired` and `Uops_Retired`) can count instructions or μ ops that are retired based on the characterization of bogus” versus non-bogus.
- **Tagging** — Tagging is a means of marking μ ops that have encountered a particular performance event so they can be counted at retirement. During the course of execution, the same event can happen more than once per μ op and a direct count of the event would not provide an indication of how many μ ops encountered that event. The tagging mechanisms allow a μ op to be tagged once during its lifetime and thus counted once at retirement. The retired suffix is used for performance metrics that increment a count once per μ op, rather than once per event. For example, a μ op may encounter a cache miss more than once during its life time, but a “Miss Retired” metric (that counts the number of retired μ ops that encountered a cache miss) will increment only once for that μ op. A “Miss Retired” metric would be useful for characterizing the performance of the cache hierarchy for a particular instruction sequence. Details of various performance metrics and how these can be constructed using the Pentium 4 and Intel Xeon processors performance events are provided in the Intel® 64 and IA-32 Architectures Optimization Reference Manual (see Section 1.4, “Related Literature”).
- **Replay** — To maximize performance for the common case, the Intel NetBurst microarchitecture aggressively schedules μ ops for execution before all the conditions for correct execution are guaranteed to be satisfied. In the event that all of these conditions are not satisfied, μ ops must be reissued. The mechanism that the Pentium

4 and Intel Xeon processors use for this reissuing of μ ops is called replay. Some examples of replay causes are cache misses, dependence violations, and unforeseen resource constraints. In normal operation, some number of replays is common and unavoidable. An excessive number of replays is an indication of a performance problem.

- **Assist** — When the hardware needs the assistance of microcode to deal with some event, the machine takes an assist. One example of this is an underflow condition in the input operands of a floating-point operation. The hardware must internally modify the format of the operands in order to perform the computation. Assists clear the entire machine of μ ops before they begin and are costly.

22.6.3.6.1 Using At-Retirement Counting

Processors based on Intel NetBurst microarchitecture allow counting both events and μ ops that encountered a specified event. For a subset of the at-retirement events, a μ op may be tagged when it encounters that event. The tagging mechanisms can be used in Interrupt-based event sampling, and a subset of these mechanisms can be used in PEBS. There are four independent tagging mechanisms, and each mechanism uses a different event to count μ ops tagged with that mechanism:

- **Front-end tagging** — This mechanism pertains to the tagging of μ ops that encountered front-end events (for example, trace cache and instruction counts) and are counted with the `Front_end_event` event.
- **Execution tagging** — This mechanism pertains to the tagging of μ ops that encountered execution events (for example, instruction types) and are counted with the `Execution_Event` event.
- **Replay tagging** — This mechanism pertains to tagging of μ ops whose retirement is replayed (for example, a cache miss) and are counted with the `Replay_event` event. Branch mispredictions are also tagged with this mechanism.
- **No tags** — This mechanism does not use tags. It uses the `Instr_retired` and the `Uops_retired` events.

Each tagging mechanism is independent from all others; that is, a μ op that has been tagged using one mechanism will not be detected with another mechanism's tagged- μ op detector. For example, if μ ops are tagged using the front-end tagging mechanisms, the `Replay_event` will not count those as tagged μ ops unless they are also tagged using the replay tagging mechanism. However, execution tags allow up to four different types of μ ops to be counted at retirement through execution tagging.

The independence of tagging mechanisms does not hold when using PEBS. When using PEBS, only one tagging mechanism should be used at a time.

Certain kinds of μ ops that cannot be tagged, including I/O, uncacheable and locked accesses, returns, and far transfers.

There are performance monitoring events that support at-retirement counting: specifically the `Front_end_event`, `Execution_event`, `Replay_event`, `Inst_retired`, and `Uops_retired` events. The following sections describe the tagging mechanisms for using these events to tag μ op and count tagged μ ops.

22.6.3.6.2 Tagging Mechanism for Front_end_event

The `Front_end_event` counts μ ops that have been tagged as encountering any of the following events:

- **μ op decode events** — Tagging μ ops for μ op decode events requires specifying bits in the ESCR associated with the performance-monitoring event, `Uop_type`.
- **Trace cache events** — Tagging μ ops for trace cache events may require specifying certain bits in the `MSR_TC_PRECISE_EVENT` MSR.

The MSRs that are supported by the front-end tagging mechanism must be set and one or both of the `NBOGUS` and `BOGUS` bits in the `Front_end_event` event mask must be set to count events. None of the events currently supported requires the use of the `MSR_TC_PRECISE_EVENT` MSR.

22.6.3.6.3 Tagging Mechanism For Execution_event

The execution tagging mechanism differs from other tagging mechanisms in how it causes tagging. One *upstream* ESCR is used to specify an event to detect and to specify a tag value (bits 5 through 8) to identify that event. A

second *downstream* ESCR is used to detect μ ops that have been tagged with that tag value identifier using `Execution_event` for the event selection.

The upstream ESCR that counts the event must have its tag enable flag (bit 4) set and must have an appropriate tag value mask entered in its tag value field. The 4-bit tag value mask specifies which of tag bits should be set for a particular μ op. The value selected for the tag value should coincide with the event mask selected in the downstream ESCR. For example, if a tag value of 1 is set, then the event mask of `NBOGUS0` should be enabled, correspondingly in the downstream ESCR. The downstream ESCR detects and counts tagged μ ops. The normal (not tag value) mask bits in the downstream ESCR specify which tag bits to count. If any one of the tag bits selected by the mask is set, the related counter is incremented by one. The tag enable and tag value bits are irrelevant for the downstream ESCR used to select the `Execution_event`.

The four separate tag bits allow the user to simultaneously but distinctly count up to four execution events at retirement. (This applies for interrupt-based event sampling. There are additional restrictions for PEBS as noted in Section 22.6.3.8.3, "Setting Up the PEBS Buffer.") It is also possible to detect or count combinations of events by setting multiple tag value bits in the upstream ESCR or multiple mask bits in the downstream ESCR. For example, use a tag value of 3H in the upstream ESCR and use `NBOGUS0/NBOGUS1` in the downstream ESCR event mask.

22.6.3.7 Tagging Mechanism for `Replay_event`

The replay mechanism enables tagging of μ ops for a subset of all replays before retirement. Use of the replay mechanism requires selecting the type of μ op that may experience the replay in the `MSR_PEBS_MATRIX_VERT` MSR and selecting the type of event in the `MSR_PEBS_ENABLE` MSR. Replay tagging must also be enabled with the `UOP_Tag` flag (bit 24) in the `MSR_PEBS_ENABLE` MSR.

The replay tags defined in Table A-5 also enable Processor Event-Based Sampling (PEBS, see Section 20.4.9). Each of these replay tags can also be used in normal sampling by not setting Bit 24 nor Bit 25 in `IA_32_PEBS_ENABLE_MSR`. Each of these metrics requires that the `Replay_Event` be used to count the tagged μ ops.

22.6.3.8 Processor Event-Based Sampling (PEBS)

The debug store (DS) mechanism in processors based on Intel NetBurst microarchitecture allow two types of information to be collected for use in debugging and tuning programs: PEBS records and BTS records. See Section 20.4.5, "Branch Trace Store (BTS)," for a description of the BTS mechanism.

PEBS permits the saving of precise architectural information associated with one or more performance events in the precise event records buffer, which is part of the DS save area (see Section 20.4.9, "BTS and DS Save Area"). To use this mechanism, a counter is configured to overflow after it has counted a preset number of events. After the counter overflows, the processor copies the current state of the general-purpose and EFLAGS registers and instruction pointer into a record in the precise event records buffer. The processor then resets the count in the performance counter and restarts the counter. When the precise event records buffer is nearly full, an interrupt is generated, allowing the precise event records to be saved. A circular buffer is not supported for precise event records.

PEBS is supported only for a subset of the at-retirement events: `Execution_event`, `Front_end_event`, and `Replay_event`. Also, PEBS can only be carried out using the one performance counter, the `MSR_IQ_COUNTER4` MSR.

In processors based on Intel Core microarchitecture, a similar PEBS mechanism is also supported using `IA32_PMC0` and `IA32_PERFVTSEL0` MSRs (See Section 22.6.2.4).

22.6.3.8.1 Detection of the Availability of the PEBS Facilities

The DS feature flag (bit 21) returned by the `CPUID` instruction indicates (when set) the availability of the DS mechanism in the processor, which supports the PEBS (and BTS) facilities. When this bit is set, the following PEBS facilities are available:

- The `PEBS_UNAVAILABLE` flag in the `IA32_MISC_ENABLE` MSR indicates (when clear) the availability of the PEBS facilities, including the `MSR_PEBS_ENABLE` MSR.
- The enable PEBS flag (bit 24) in the `MSR_PEBS_ENABLE` MSR allows PEBS to be enabled (set) or disabled (clear).

- The IA32_DS_AREA MSR can be programmed to point to the DS save area.

22.6.3.8.2 Setting Up the DS Save Area

Section 20.4.9.2, “Setting Up the DS Save Area,” describes how to set up and enable the DS save area. This procedure is common for PEBS and BTS.

22.6.3.8.3 Setting Up the PEBS Buffer

Only the MSR_IQ_COUNTER4 performance counter can be used for PEBS. Use the following procedure to set up the processor and this counter for PEBS:

1. Set up the precise event buffering facilities. Place values in the precise event buffer base, precise event index, precise event absolute maximum, and precise event interrupt threshold, and precise event counter reset fields of the DS buffer management area (see Figure 20-5) to set up the precise event records buffer in memory.
2. Enable PEBS. Set the Enable PEBS flag (bit 24) in MSR_PEBS_ENABLE MSR.
3. Set up the MSR_IQ_COUNTER4 performance counter and its associated CCCR and one or more ESCRs for PEBS.

22.6.3.8.4 Writing a PEBS Interrupt Service Routine

The PEBS facilities share the same interrupt vector and interrupt service routine (called the DS ISR) with the non-precise event-based sampling and BTS facilities. To handle PEBS interrupts, PEBS handler code must be included in the DS ISR. See Section 20.4.9.5, “Writing the DS Interrupt Service Routine,” for guidelines for writing the DS ISR.

22.6.3.8.5 Other DS Mechanism Implications

The DS mechanism is not available in the SMM. It is disabled on transition to the SMM mode. Similarly the DS mechanism is disabled on the generation of a machine check exception and is cleared on processor RESET and INIT.

The DS mechanism is available in real address mode.

22.6.3.9 Operating System Implications

The DS mechanism can be used by the operating system as a debugging extension to facilitate failure analysis. When using this facility, a 25 to 30 times slowdown can be expected due to the effects of the trace store occurring on every taken branch.

Depending upon intended usage, the instruction pointers that are part of the branch records or the PEBS records need to have an association with the corresponding process. One solution requires the ability for the DS specific operating system module to be chained to the context switch. A separate buffer can then be maintained for each process of interest and the MSR pointing to the configuration area saved and setup appropriately on each context switch.

If the BTS facility has been enabled, then it must be disabled and state stored on transition of the system to a sleep state in which processor context is lost. The state must be restored on return from the sleep state.

It is required that an interrupt gate be used for the DS interrupt as opposed to a trap gate to prevent the generation of an endless interrupt loop.

Pages that contain buffers must have mappings to the same physical address for all processes/logical processors, such that any change to CR3 will not change DS addresses. If this requirement cannot be satisfied (that is, the feature is enabled on a per thread/process basis), then the operating system must ensure that the feature is enabled/disabled appropriately in the context switch code.

22.6.4 Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture

The performance monitoring capability of processors based on Intel NetBurst microarchitecture and supporting Intel Hyper-Threading Technology is similar to that described in Section 22.6.3. However, the capability is extended so that:

- Performance counters can be programmed to select events qualified by logical processor IDs.
- Performance monitoring interrupts can be directed to a specific logical processor within the physical processor.

The sections below describe performance counters, event qualification by logical processor ID, and special purpose bits in ESCRs/CCCRs. They also describe MSR_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT, and MSR_TC_PRECISE_EVENT.

22.6.4.1 ESCR MSRs

Figure 22-53 shows the layout of an ESCR MSR in processors supporting Intel Hyper-Threading Technology.

The functions of the flags and fields are as follows:

- **T1_USR flag, bit 0** — When set, events are counted when thread 1 (logical processor 1) is executing at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.

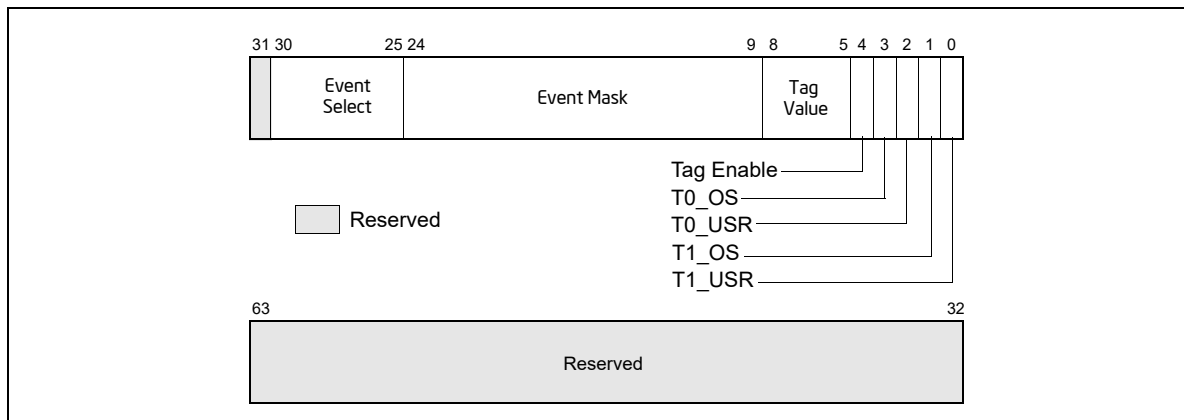


Figure 22-53. Event Selection Control Register (ESCR) for the Pentium 4 Processor, Intel® Xeon® Processor, and Intel® Xeon® Processor MP Supporting Hyper-Threading Technology

- **T1_OS flag, bit 1** — When set, events are counted when thread 1 (logical processor 1) is executing at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the T1_OS and T1_USR flags are set, thread 1 events are counted at all privilege levels.)
- **T0_USR flag, bit 2** — When set, events are counted when thread 0 (logical processor 0) is executing at a CPL of 1, 2, or 3.
- **T0_OS flag, bit 3** — When set, events are counted when thread 0 (logical processor 0) is executing at CPL of 0. (When both the T0_OS and T0_USR flags are set, thread 0 events are counted at all privilege levels.)
- **Tag enable, bit 4** — When set, enables tagging of μ ops to assist in at-retirement event counting; when clear, disables tagging. See Section 22.6.3.6, "At-Retirement Counting."
- **Tag value field, bits 5 through 8** — Selects a tag value to associate with a μ op to assist in at-retirement event counting.
- **Event mask field, bits 9 through 24** — Selects events to be counted from the event class selected with the event select field.
- **Event select field, bits 25 through 30** — Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

The T0_OS and T0_USR flags and the T1_OS and T1_USR flags allow event counting and sampling to be specified for a specific logical processor (0 or 1) within an Intel Xeon processor MP (See also: Section 11.4.5, “Identifying Logical Processors in an MP System,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A).

Not all performance monitoring events can be detected within an Intel Xeon processor MP on a per logical processor basis (see Section 22.6.4.4, “Performance Monitoring Events”). Some sub-events (specified by an event mask bits) are counted or sampled without regard to which logical processor is associated with the detected event.

22.6.4.2 CCCR MSRs

Figure 22-54 shows the layout of a CCCR MSR in processors supporting Intel Hyper-Threading Technology. The functions of the flags and fields are as follows:

- **Enable flag, bit 12** — When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset
 - **ESCR select field, bits 13 through 15** — Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.
 - **Active thread field, bits 16 and 17** — Enables counting depending on which logical processors are active (executing a thread). This field enables filtering of events based on the state (active or inactive) of the logical processors. The encodings of this field are as follows:
 - 00** — None. Count only when neither logical processor is active.
 - 01** — Single. Count only when one logical processor is active (either 0 or 1).
 - 10** — Both. Count only when both logical processors are active.
 - 11** — Any. Count when either logical processor is active.
- A halted logical processor or a logical processor in the “wait for SIPI” state is considered inactive.
- **Compare flag, bit 18** — When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.

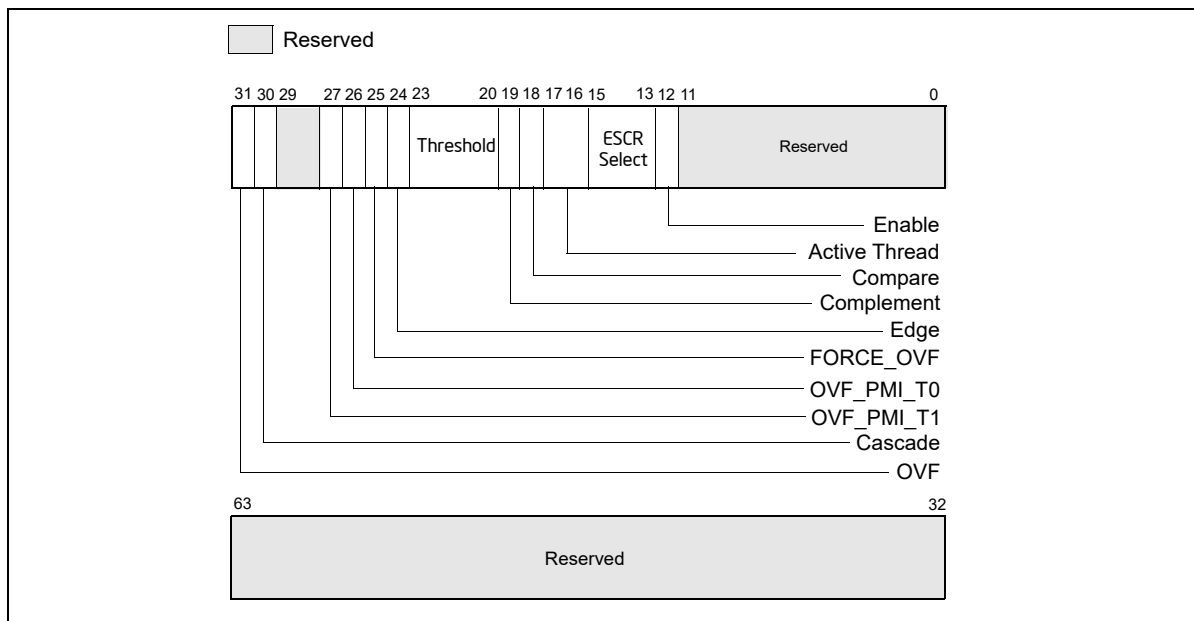


Figure 22-54. Counter Configuration Control Register (CCCR)

- **Complement flag, bit 19** — Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance counter; when clear, counts greater than the threshold value result in a count being

delivered to the performance counter (see Section 22.6.3.5.2, “Filtering Events”). The compare flag is not active unless the compare flag is set.

- **Threshold field, bits 20 through 23** — Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 22.6.3.5.2, “Filtering Events”).
- **Edge flag, bit 24** — When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.
- **FORCE_OVF flag, bit 25** — When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.
- **OVF_PMI_T0 flag, bit 26** — When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 0 when the counter overflows occurs; when clear, disables PMI generation for logical processor 0. Note that the PMI is generate on the next event count after the counter has overflowed.
- **OVF_PMI_T1 flag, bit 27** — When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 1 when the counter overflows occurs; when clear, disables PMI generation for logical processor 1. Note that the PMI is generate on the next event count after the counter has overflowed.
- **Cascade flag, bit 30** — When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 22.6.3.2, “Performance Counters,” for further details); when clear, disables cascading of counters.
- **OVF flag, bit 31** — Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

22.6.4.3 IA32_PEBS_ENABLE MSR

In a processor supporting Intel Hyper-Threading Technology and based on the Intel NetBurst microarchitecture, PEBS is enabled and qualified with two bits in the MSR_PEBS_ENABLE MSR: bit 25 (ENABLE_PEBS_MY_THR) and 26 (ENABLE_PEBS_OTH_THR) respectively. These bits do not explicitly identify a specific logical processor by logic processor ID(T0 or T1); instead, they allow a software agent to enable PEBS for subsequent threads of execution on the same logical processor on which the agent is running (“my thread”) or for the other logical processor in the physical package on which the agent is not running (“other thread”).

PEBS is supported for only a subset of the at-retirement events: Execution_event, Front_end_event, and Replay_event. Also, PEBS can be carried out only with two performance counters: MSR_IQ_CCCR4 (MSR address 370H) for logical processor 0 and MSR_IQ_CCCR5 (MSR address 371H) for logical processor 1.

Performance monitoring tools should use a processor affinity mask to bind the kernel mode components that need to modify the ENABLE_PEBS_MY_THR and ENABLE_PEBS_OTH_THR bits in the MSR_PEBS_ENABLE MSR to a specific logical processor. This is to prevent these kernel mode components from migrating between different logical processors due to OS scheduling.

22.6.4.4 Performance Monitoring Events

When Intel Hyper-Threading Technology is active, many performance monitoring events can be can be qualified by the logical processor ID, which corresponds to bit 0 of the initial APIC ID. This allows for counting an event in any or all of the logical processors. However, not all the events have this logic processor specificity, or thread specificity.

Here, each event falls into one of two categories:

- **Thread specific (TS)** — The event can be qualified as occurring on a specific logical processor.
- **Thread independent (TI)** — The event cannot be qualified as being associated with a specific logical processor.

If for example, a TS event occurred in logical processor T0, the counting of the event (as shown in Table 22-93) depends only on the setting of the T0_USR and T0_OS flags in the ESCR being used to set up the event counter. The T1_USR and T1_OS flags have no effect on the count.

Table 22-93. Effect of Logical Processor and CPL Qualification for Logical-Processor-Specific (TS) Events

	T1_OS/T1_USR = 00	T1_OS/T1_USR = 01	T1_OS/T1_USR = 11	T1_OS/T1_USR = 10
T0_OS/T0_USR = 00	Zero count	Counts while T1 in USR	Counts while T1 in OS or USR	Counts while T1 in OS
T0_OS/T0_USR = 01	Counts while T0 in USR	Counts while T0 in USR or T1 in USR	Counts while (a) T0 in USR or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR = 11	Counts while T0 in OS or USR	Counts while (a) T0 in OS or (b) T0 in USR or (c) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T0 in USR or (c) T1 in OS
T0_OS/T0_USR = 10	Counts T0 in OS	Counts T0 in OS or T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS

When a bit in the event mask field is TI, the effect of specifying bit-0-3 of the associated ESCR are described in Table 15-6. For events that are marked as TI, the effect of selectively specifying T0_USR, T0_OS, T1_USR, T1_OS bits is shown in Table 22-94.

Table 22-94. Effect of Logical Processor and CPL Qualification for Non-logical-Processor-specific (TI) Events

	T1_OS/T1_USR = 00	T1_OS/T1_USR = 01	T1_OS/T1_USR = 11	T1_OS/T1_USR = 10
T0_OS/T0_USR = 00	Zero count	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR = 01	Counts while (a) T0 in USR or (b) T1 in USR	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR = 11	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR = 0	Counts while (a) T0 in OS or (b) T1 in OS	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS

22.6.4.5 Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture

22.6.4.5.1 Non-Halted Clockticks

Use the following procedure to program ESCRs and CCCRs to obtain non-halted clockticks on processors based on Intel NetBurst microarchitecture:

1. Select an ESCR for the global_power_events and specify the RUNNING sub-event mask and the desired T0_OS/T0_USR/T1_OS/T1_USR bits for the targeted processor.
2. Select an appropriate counter.
3. Enable counting in the CCCR for that counter by setting the enable bit.

22.6.4.5.2 Non-Sleep Clockticks

Performance monitoring counters can be configured to count clockticks whenever the performance monitoring hardware is not powered-down. To count Non-sleep Clockticks with a performance-monitoring counter, do the following:

1. Select one of the 18 counters.
2. Select any of the ESCRs whose events the selected counter can count. Set its event select to anything other than "no_event"; the counter may be disabled if this is not done.

3. Turn threshold comparison on in the CCCR by setting the compare bit to “1”.
4. Set the threshold to “15” and the complement to “1” in the CCCR. Since no event can exceed this threshold, the threshold condition is met every cycle and the counter counts every cycle. Note that this overrides any qualification (e.g., by CPL) specified in the ESCR.
5. Enable counting in the CCCR for the counter by setting the enable bit.

In most cases, the counts produced by the non-halted and non-sleep metrics are equivalent if the physical package supports one logical processor and is not placed in a power-saving state. Operating systems may execute an HLT instruction and place a physical processor in a power-saving state.

On processors that support Intel Hyper-Threading Technology (Intel HT Technology), each physical package can support two or more logical processors. Current implementation of Intel HT Technology provides two logical processors for each physical processor. While both logical processors can execute two threads simultaneously, one logical processor may halt to allow the other logical processor to execute without sharing execution resources between two logical processors.

Non-halted Clockticks can be set up to count the number of processor clock cycles for each logical processor whenever the logical processor is not halted (the count may include some portion of the clock cycles for that logical processor to complete a transition to a halted state). Physical processors that support Intel HT Technology enter into a power-saving state if all logical processors halt.

The Non-sleep Clockticks mechanism uses a filtering mechanism in CCCRs. The mechanism will continue to increment as long as one logical processor is not halted or in a power-saving state. Applications may cause a processor to enter into a power-saving state by using an OS service that transfers control to an OS's idle loop. The idle loop then may place the processor into a power-saving state after an implementation-dependent period if there is no work for the processor.

22.6.5 Performance Monitoring and Dual-Core Technology

The performance monitoring capability of dual-core processors duplicates the microarchitectural resources of a single-core processor implementation. Each processor core has dedicated performance monitoring resources.

In the case of Pentium D processor, each logical processor is associated with dedicated resources for performance monitoring. In the case of Pentium processor Extreme edition, each processor core has dedicated resources, but two logical processors in the same core share performance monitoring resources (see Section 22.6.4, “Performance Monitoring and Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture”).

22.6.6 Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache

The 64-bit Intel Xeon processor MP with up to 8-MByte L3 cache has a CUID signature of family [0FH], model [03H or 04H]. Performance monitoring capabilities available to Pentium 4 and Intel Xeon processors with the same values (see Section 22.1 and Section 22.6.4) apply to the 64-bit Intel Xeon processor MP with an L3 cache.

The level 3 cache is connected between the system bus and IOQ through additional control logic. See Figure 22-55.

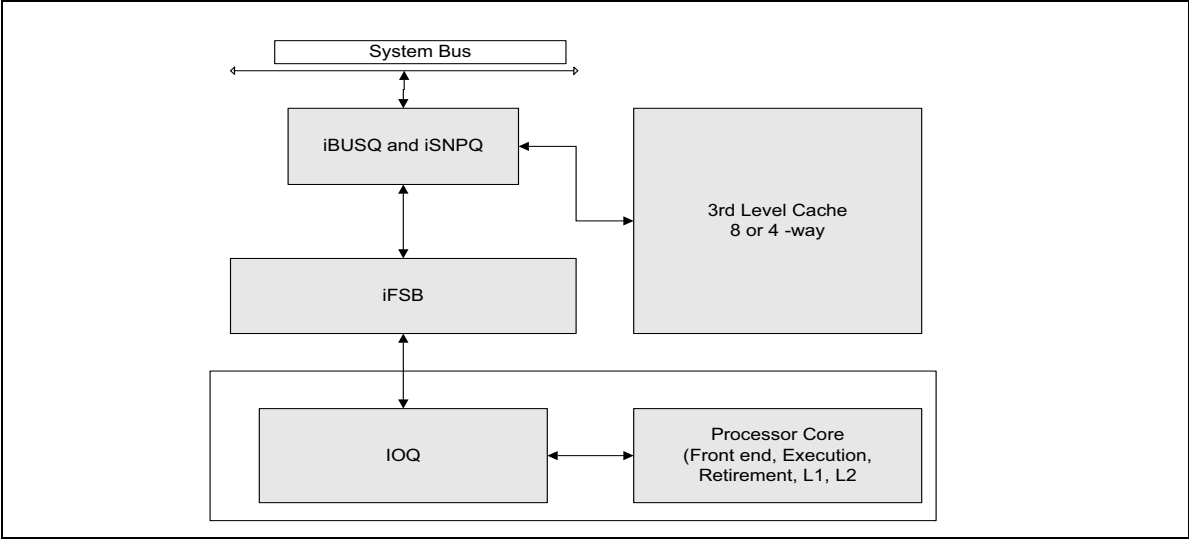


Figure 22-55. Block Diagram of 64-bit Intel® Xeon® Processor MP with 8-MByte L3

Additional performance monitoring capabilities and facilities unique to 64-bit Intel Xeon processor MP with an L3 cache are described in this section. The facility for monitoring events consists of a set of dedicated model-specific registers (MSRs), each dedicated to a specific event. Programming of these MSRs requires using RDMSR/WRMSR instructions with 64-bit values.

The lower 32-bits of the MSRs at addresses 107CC through 107D3 are treated as 32 bit performance counter registers. These performance counters can be accessed using RDPMC instruction with the index starting from 18 through 25. The EDX register returns zero when reading these 8 PMCs.

The performance monitoring capabilities consist of four events. These are:

- **IBUSQ event** — This event detects the occurrence of micro-architectural conditions related to the iBUSQ unit. It provides two MSRs: MSR_IFSB_IBUSQ0 and MSR_IFSB_IBUSQ1. Configure sub-event qualification and enable/disable functions using the high 32 bits of these MSRs. The low 32 bits act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the upper 32 bits. See Figure 22-56.

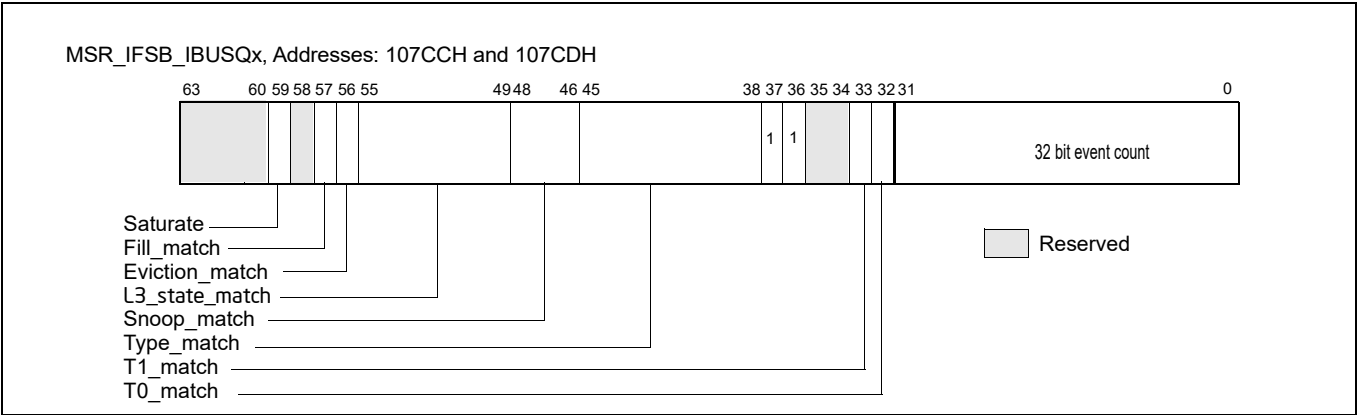


Figure 22-56. MSR_IFSB_IBUSQx, Addresses: 107CCH and 107CDH

- **ISNPQ event** — This event detects the occurrence of microarchitectural conditions related to the iSNPQ unit. It provides two MSRs: MSR_IFSB_ISNPQ0 and MSR_IFSB_ISNPQ1. Configure sub-event qualifications and enable/disable functions using the high 32 bits of the MSRs. The low 32-bits act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the upper 32-bits. See Figure 22-57.

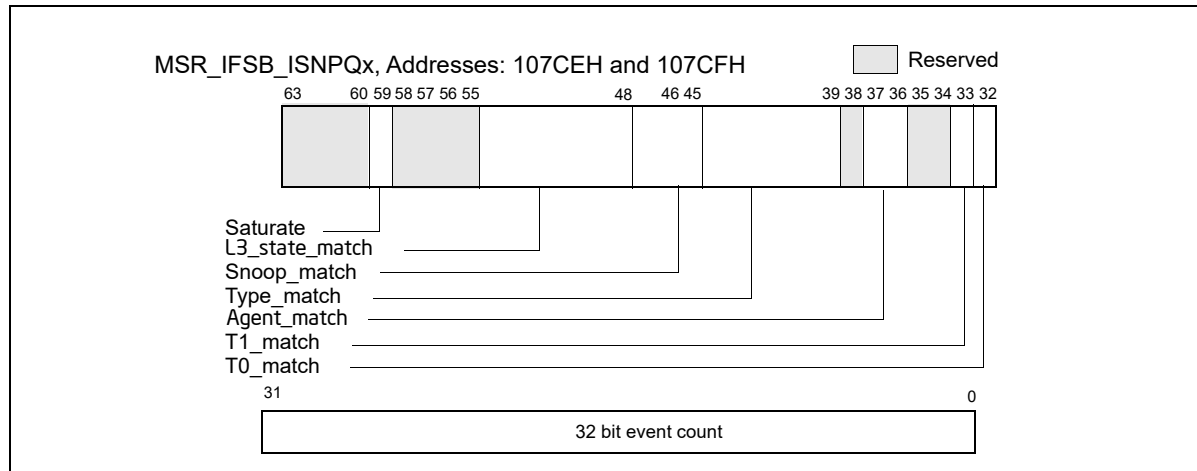


Figure 22-57. MSR_IFSB_ISNPQx, Addresses: 107CEH and 107CFH

- EFSB event** — This event can detect the occurrence of micro-architectural conditions related to the iFSB unit or system bus. It provides two MSRs: MSR_EFSB_DRDY0 and MSR_EFSB_DRDY1. Configure sub-event qualifications and enable/disable functions using the high 32 bits of the 64-bit MSR. The low 32-bit act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the qualification bits in the upper 32-bits of the MSR. See Figure 22-58.

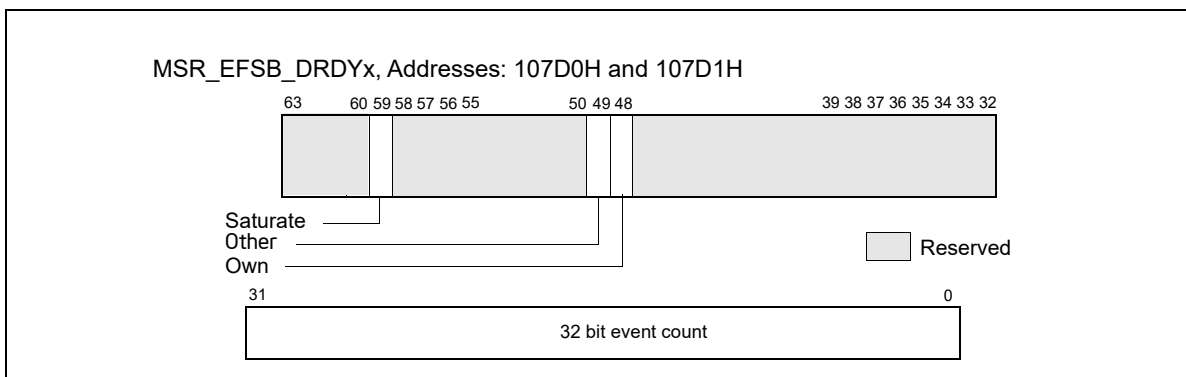


Figure 22-58. MSR_EFSB_DRDYx, Addresses: 107D0H and 107D1H

- IBUSQ Latency event** — This event accumulates weighted cycle counts for latency measurement of transactions in the iBUSQ unit. The count is enabled by setting MSR_IFSB_CTRL6[bit 26] to 1; the count freezes after software sets MSR_IFSB_CTRL6[bit 26] to 0. MSR_IFSB_CNTR7 acts as a 64-bit event counter for this event. See Figure 22-59.

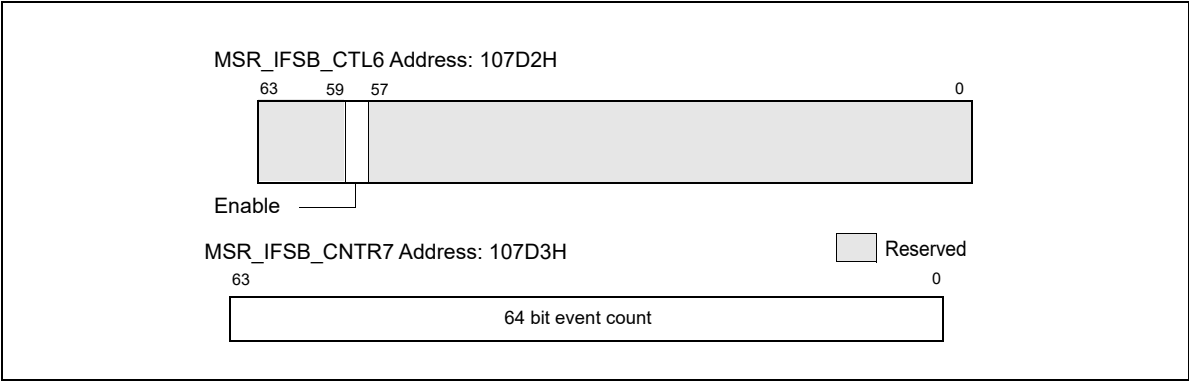


Figure 22-59. MSR_IFSB_CTL6, Address: 107D2H; MSR_IFSB_CNTR7, Address: 107D3H

22.6.7 Performance Monitoring on L3 and Caching Bus Controller Sub-Systems

The Intel Xeon processor 7400 series and Dual-Core Intel Xeon processor 7100 series employ a distinct L3/caching bus controller sub-system. These sub-system have a unique set of performance monitoring capability and programming interfaces that are largely common between these two processor families.

Intel Xeon processor 7400 series are based on 45 nm enhanced Intel Core microarchitecture. The CUID signature is indicated by DisplayFamily_DisplayModel value of 06_1DH (see the CUID instruction in Chapter 3, “Instruction Set Reference, A-L,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A). Intel Xeon processor 7400 series have six processor cores that share an L3 cache.

Dual-Core Intel Xeon processor 7100 series are based on Intel NetBurst microarchitecture, have a CUID signature of family [0FH], model [06H] and a unified L3 cache shared between two cores. Each core in an Intel Xeon processor 7100 series supports Intel Hyper-Threading Technology, providing two logical processors per core.

Both Intel Xeon processor 7400 series and Intel Xeon processor 7100 series support multi-processor configurations using system bus interfaces. In Intel Xeon processor 7400 series, the L3/caching bus controller sub-system provides three Simple Direct Interface (SDI) to service transactions originated the XQ-replacement SDI logic in each dual-core modules. In Intel Xeon processor 7100 series, the IOQ logic in each processor core is replaced with a Simple Direct Interface (SDI) logic. The L3 cache is connected between the system bus and the SDI through additional control logic. See Figure 22-60 for the block configuration of six processor cores and the L3/Caching bus controller sub-system in Intel Xeon processor 7400 series. Figure 22-60 shows the block configuration of two processor cores (four logical processors) and the L3/Caching bus controller sub-system in Intel Xeon processor 7100 series.

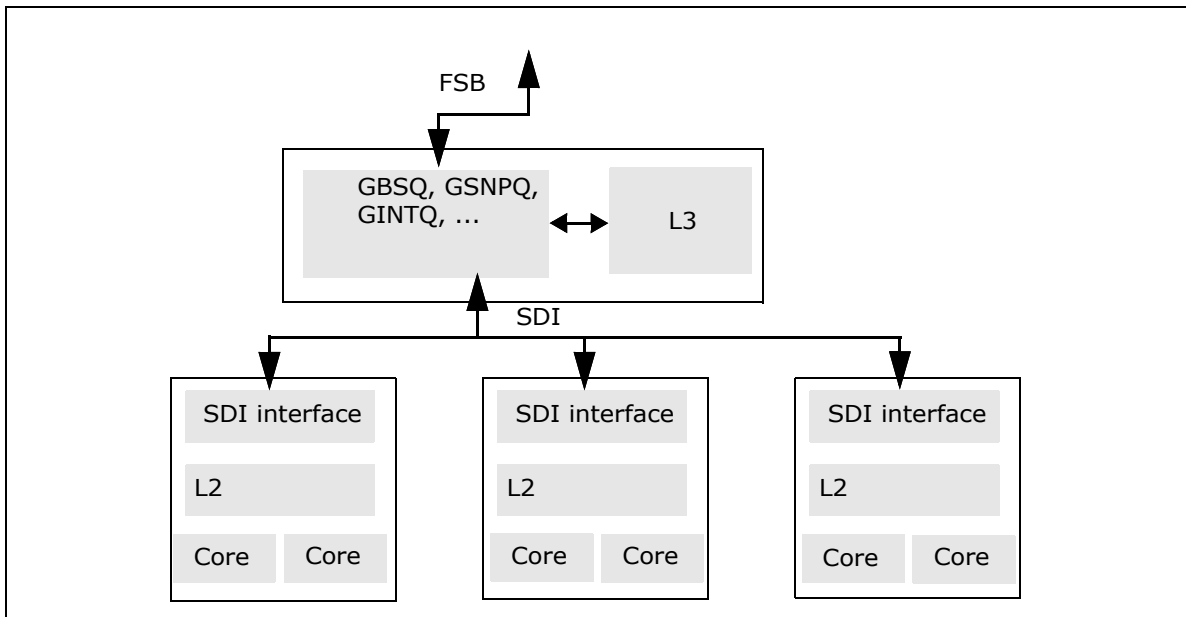


Figure 22-60. Block Diagram of the Intel® Xeon® Processor 7400 Series

Almost all of the performance monitoring capabilities available to processor cores with the same CPUID signatures (see Section 22.1 and Section 22.6.4) apply to Intel Xeon processor 7100 series. The MSRs used by performance monitoring interface are shared between two logical processors in the same processor core.

The performance monitoring capabilities available to processor with DisplayFamily_DisplayModel signature 06_17H also apply to Intel Xeon processor 7400 series. Each processor core provides its own set of MSRs for performance monitoring interface.

The IOQ_allocation and IOQ_active_entries events are not supported in Intel Xeon processor 7100 series and 7400 series. Additional performance monitoring capabilities applicable to the L3/caching bus controller sub-system are described in this section.

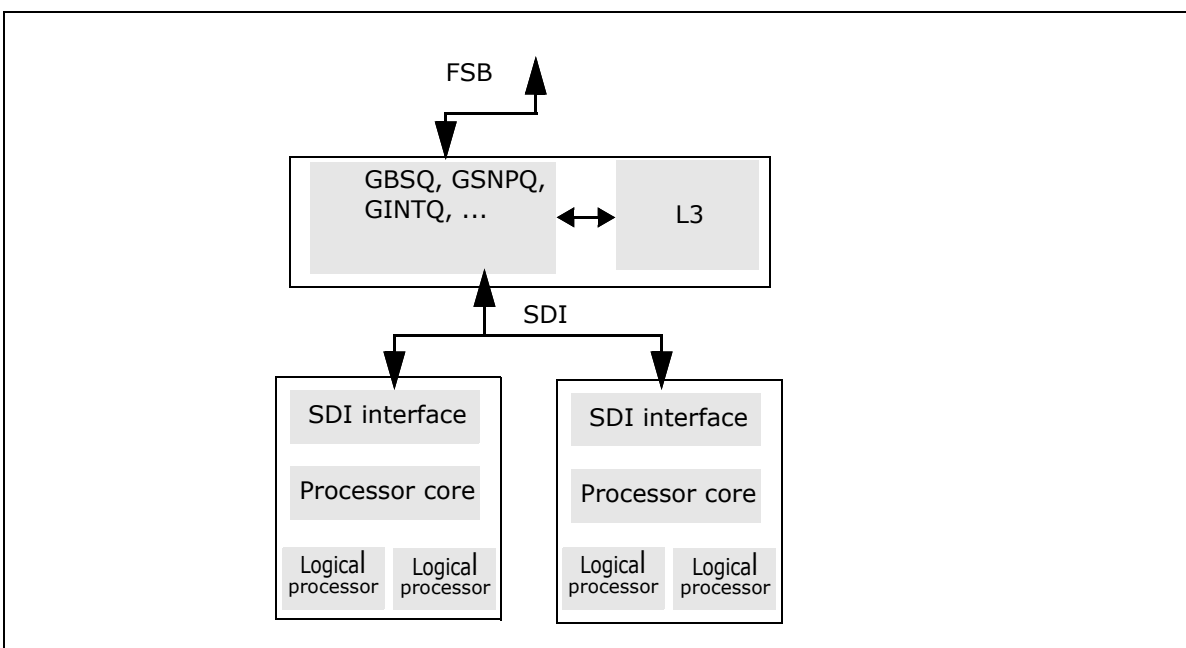


Figure 22-61. Block Diagram of the Intel® Xeon® Processor 7100 Series

22.6.7.1 Overview of Performance Monitoring with L3/Caching Bus Controller

The facility for monitoring events consists of a set of dedicated model-specific registers (MSRs). There are eight event select/counting MSRs that are dedicated to counting events associated with specified microarchitectural conditions. Programming of these MSRs requires using RDMSR/WRMSR instructions with 64-bit values. In addition, an MSR MSR_EMON_L3_GL_CTL provides simplified interface to control freezing, resetting, re-enabling operation of any combination of these event select/counting MSRs.

The eight MSRs dedicated to count occurrences of specific conditions are further divided to count three sub-classes of microarchitectural conditions:

- Two MSRs (MSR_EMON_L3_CTR_CTL0 and MSR_EMON_L3_CTR_CTL1) are dedicated to counting GBSQ events. Up to two GBSQ events can be programmed and counted simultaneously.
- Two MSRs (MSR_EMON_L3_CTR_CTL2 and MSR_EMON_L3_CTR_CTL3) are dedicated to counting GSNPQ events. Up to two GBSQ events can be programmed and counted simultaneously.
- Four MSRs (MSR_EMON_L3_CTR_CTL4, MSR_EMON_L3_CTR_CTL5, MSR_EMON_L3_CTR_CTL6, and MSR_EMON_L3_CTR_CTL7) are dedicated to counting external bus operations.

The bit fields in each of eight MSRs share the following common characteristics:

- Bits 63:32 is the event control field that includes an event mask and other bit fields that control counter operation. The event mask field specifies details of the microarchitectural condition, and its definition differs across GBSQ, GSNPQ, FSB.
- Bits 31:0 is the event count field. If the specified condition is met during each relevant clock domain of the event logic, the matched condition signals the counter logic to increment the associated event count field. The lower 32-bits of these 8 MSRs at addresses 107CC through 107D3 are treated as 32 bit performance counter registers.

In Dual-Core Intel Xeon processor 7100 series, the uncore performance counters can be accessed using RDPMC instruction with the index starting from 18 through 25. The EDX register returns zero when reading these 8 PMCs.

In Intel Xeon processor 7400 series, RDPMC with ECX between 2 and 9 can be used to access the eight uncore performance counter/control registers.

22.6.7.2 GBSQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL0 and MSR_EMON_L3_CTR_CTL1 is given in Figure 22-62. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) consists of the following eight attributes:

- Agent_Select (bits 35:32): The definition of this field differs slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series, each bit specifies a logical processor in the physical package. The lower two bits corresponds to two logical processors in the first processor core, the upper two bits corresponds to two logical processors in the second processor core. 0FH encoding matches transactions from any logical processor.

For Intel Xeon processor 7400 series, each bit of [34:32] specifies the SDI logic of a dual-core module as the originator of the transaction. A value of 0111B in bits [35:32] specifies transaction from any processor core.

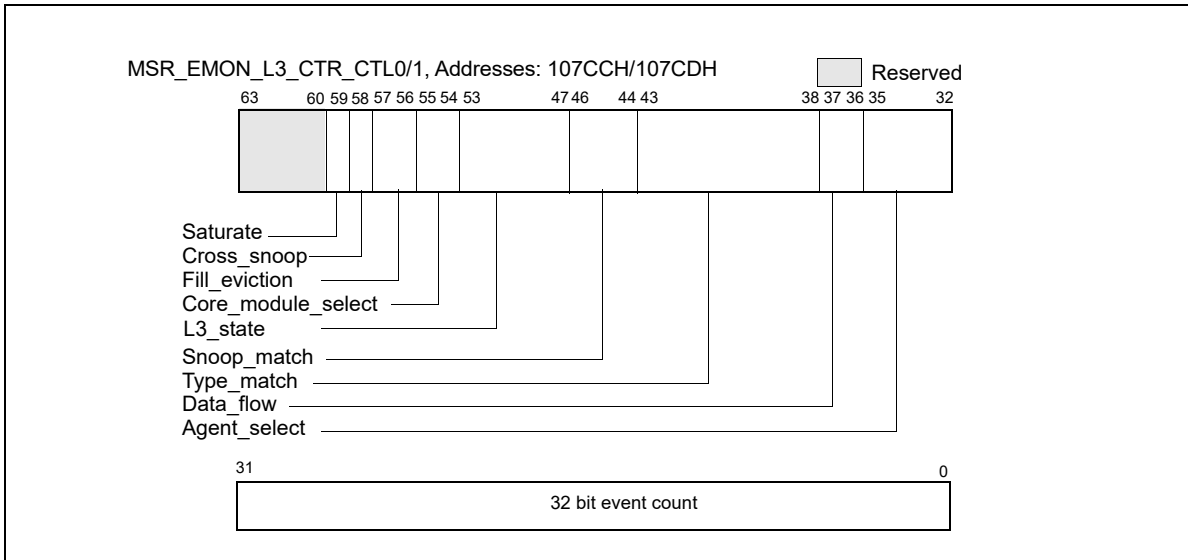


Figure 22-62. MSR_EMON_L3_CTL0/1, Addresses: 107CCH/107CDH

- Data_Flow (bits 37:36): Bit 36 specifies demand transactions, bit 37 specifies prefetch transactions.
- Type_Match (bits 43:38): Specifies transaction types. If all six bits are set, event count will include all transaction types.
- Snoop_Match: (bits 46:44): The three bits specify (in ascending bit position) clean snoop result, HIT snoop result, and HITM snoop results respectively.
- L3_State (bits 53:47): Each bit specifies an L2 coherency state.
- Core_Module_Select (bits 55:54): The valid encodings for L3 lookup differ slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series,

- 00B: Match transactions from any core in the physical package
- 01B: Match transactions from this core only
- 10B: Match transactions from the other core in the physical package
- 11B: Match transaction from both cores in the physical package

For Intel Xeon processor 7400 series,

- 00B: Match transactions from any dual-core module in the physical package
- 01B: Match transactions from this dual-core module only
- 10B: Match transactions from either one of the other two dual-core modules in the physical package
- 11B: Match transaction from more than one dual-core modules in the physical package

- Fill_Eviction (bits 57:56): The valid encodings are
 - 00B: Match any transactions
 - 01B: Match transactions that fill L3
 - 10B: Match transactions that fill L3 without an eviction
 - 11B: Match transaction fill L3 with an eviction
- Cross_Snoop (bit 58): The encodings are
 - 0B: Match any transactions
 - 1B: Match cross snoop transactions

For each counting clock domain, if all eight attributes match, event logic signals to increment the event count field.

22.6.7.3 GSNPQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL2 and MSR_EMON_L3_CTR_CTL3 is given in Figure 22-63. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) consists of the following six attributes:

- **Agent_Select** (bits 37:32): The definition of this field differs slightly between Intel Xeon processor 7100 and 7400.
- For Intel Xeon processor 7100 series, each of the lowest 4 bits specifies a logical processor in the physical package. The lowest two bits corresponds to two logical processors in the first processor core, the next two bits corresponds to two logical processors in the second processor core. Bit 36 specifies other symmetric agent transactions. Bit 37 specifies central agent transactions. 3FH encoding matches transactions from any logical processor.

For Intel Xeon processor 7400 series, each of the lowest 3 bits specifies a dual-core module in the physical package. Bit 37 specifies central agent transactions.

- **Type_Match** (bits 43:38): Specifies transaction types. If all six bits are set, event count will include any transaction types.
- **Snoop_Match**: (bits 46:44): The three bits specify (in ascending bit position) clean snoop result, HIT snoop result, and HITM snoop results respectively.
- **L2_State** (bits 53:47): Each bit specifies an L3 coherency state.
- **Core_Module_Select** (bits 56:54): Bit 56 enables Core_Module_Select matching. If bit 56 is clear, Core_Module_Select encoding is ignored. The valid encodings for the lower two bits (bit 55, 54) differ slightly between Intel Xeon processor 7100 and 7400.

For Intel Xeon processor 7100 series, if bit 56 is set, the valid encodings for the lower two bits (bit 55, 54) are

- 00B: Match transactions from only one core (irrespective which core) in the physical package
- 01B: Match transactions from this core and not the other core
- 10B: Match transactions from the other core in the physical package, but not this core
- 11B: Match transaction from both cores in the physical package

For Intel Xeon processor 7400 series, if bit 56 is set, the valid encodings for the lower two bits (bit 55, 54) are

- 00B: Match transactions from only one dual-core module (irrespective which module) in the physical package.
- 01B: Match transactions from one or more dual-core modules.
- 10B: Match transactions from two or more dual-core modules.
- 11B: Match transaction from all three dual-core modules in the physical package.

- **Block_Snoop** (bit 57): specifies blocked snoop.

For each counting clock domain, if all six attributes match, event logic signals to increment the event count field.

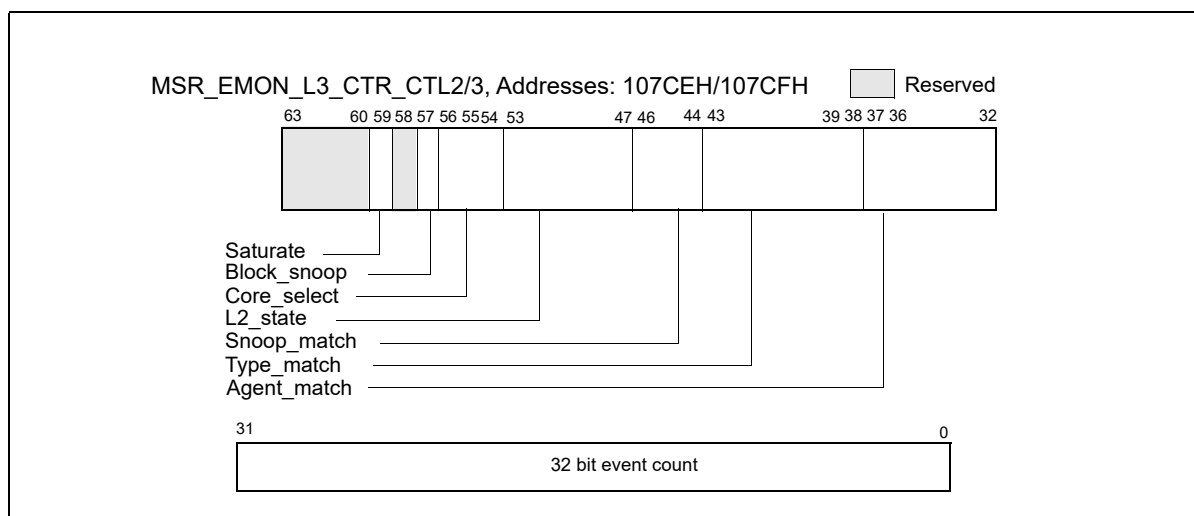


Figure 22-63. MSR_EMON_L3_CTR_CTL2/3, Addresses: 107CEH/107CFH

22.6.7.4 FSB Event Interface

The layout of MSR_EMON_L3_CTR_CTL4 through MSR_EMON_L3_CTR_CTL7 is given in Figure 22-64. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) is organized as follows:

- Bit 58: must set to 1.
- FSB_Submask (bits 57:32): Specifies FSB-specific sub-event mask.

The FSB sub-event mask defines a set of independent attributes. The event logic signals to increment the associated event count field if one of the attribute matches. Some of the sub-event mask bit counts durations. A duration event increments at most once per cycle.

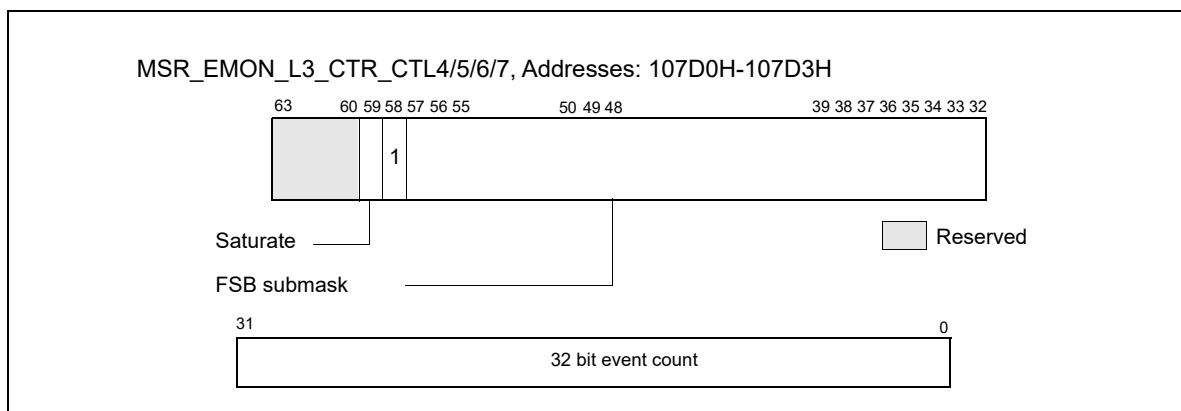


Figure 22-64. MSR_EMON_L3_CTR_CTL4/5/6/7, Addresses: 107D0H-107D3H

22.6.7.4.1 FSB Sub-Event Mask Interface

- FSB_type (bit 37:32): Specifies different FSB transaction types originated from this physical package.
- FSB_L_clear (bit 38): Count clean snoop results from any source for transaction originated from this physical package.
- FSB_L_hit (bit 39): Count HIT snoop results from any source for transaction originated from this physical package.

- FSB_L_hitm (bit 40): Count HITM snoop results from any source for transaction originated from this physical package.
- FSB_L_defer (bit 41): Count DEFER responses to this processor's transactions.
- FSB_L_retry (bit 42): Count RETRY responses to this processor's transactions.
- FSB_L_snoop_stall (bit 43): Count snoop stalls to this processor's transactions.
- FSB_DBSY (bit 44): Count DBSY assertions by this processor (without a concurrent DRDY).
- FSB_DRDY (bit 45): Count DRDY assertions by this processor.
- FSB_BNR (bit 46): Count BNR assertions by this processor.
- FSB_IOQ_empty (bit 47): Counts each bus clocks when the IOQ is empty.
- FSB_IOQ_full (bit 48): Counts each bus clocks when the IOQ is full.
- FSB_IOQ_active (bit 49): Counts each bus clocks when there is at least one entry in the IOQ.
- FSB_WW_data (bit 50): Counts back-to-back write transaction's data phase.
- FSB_WW_issue (bit 51): Counts back-to-back write transaction request pairs issued by this processor.
- FSB_WR_issue (bit 52): Counts back-to-back write-read transaction request pairs issued by this processor.
- FSB_RW_issue (bit 53): Counts back-to-back read-write transaction request pairs issued by this processor.
- FSB_other_DBSY (bit 54): Count DBSY assertions by another agent (without a concurrent DRDY).
- FSB_other_DRDY (bit 55): Count DRDY assertions by another agent.
- FSB_other_snoop_stall (bit 56): Count snoop stalls on the FSB due to another agent.
- FSB_other_BNR (bit 57): Count BNR assertions from another agent.

22.6.7.5 Common Event Control Interface

The MSR_EMON_L3_GL_CTL MSR provides simplified access to query overflow status of the GBSQ, GSNPQ, FSB event counters. It also provides control bit fields to freeze, unfreeze, or reset those counters. The following bit fields are supported:

- GL_freeze_cmd (bit 0): Freeze the event counters specified by the GL_event_select field.
- GL_unfreeze_cmd (bit 1): Unfreeze the event counters specified by the GL_event_select field.
- GL_reset_cmd (bit 2): Clear the event count field of the event counters specified by the GL_event_select field. The event select field is not affected.
- GL_event_select (bit 23:16): Selects one or more event counters to subject to specified command operations indicated by bits 2:0. Bit 16 corresponds to MSR_EMON_L3_CTR_CTL0, bit 23 corresponds to MSR_EMON_L3_CTR_CTL7.
- GL_event_status (bit 55:48): Indicates the overflow status of each event counters. Bit 48 corresponds to MSR_EMON_L3_CTR_CTL0, bit 55 corresponds to MSR_EMON_L3_CTR_CTL7.

In the event control field (bits 63:32) of each MSR, if the saturate control (bit 59, see Figure 22-62 for example) is set, the event logic forces the value FFFF_FFFFH into the event count field instead of incrementing it.

22.6.8 Performance Monitoring (P6 Family Processor)

The P6 family processors provide two 40-bit performance counters, allowing two types of events to be monitored simultaneously. These can either count events or measure duration. When counting events, a counter increments each time a specified event takes place or a specified number of events takes place. When measuring duration, it counts the number of processor clocks that occur while a specified condition is true. The counters can count events or measure durations that occur at any privilege level.

NOTE

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. Counter values reported are not guaranteed to be accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The performance-monitoring counters are supported by four MSRs: the performance event select MSRs (PerfEvtSel0 and PerfEvtSel1) and the performance counter MSRs (PerfCtr0 and PerfCtr1). These registers can be read from and written to using the RDMSR and WRMSR instructions, respectively. They can be accessed using these instructions only when operating at privilege level 0. The PerfCtr0 and PerfCtr1 MSRs can be read from any privilege level using the RDPNC (read performance-monitoring counters) instruction.

NOTE

The PerfEvtSel0, PerfEvtSel1, PerfCtr0, and PerfCtr1 MSRs and the events listed for P6 family processors are model-specific for P6 family processors. They are not guaranteed to be available in other IA-32 processors.

22.6.8.1 PerfEvtSel0 and PerfEvtSel1 MSRs

The PerfEvtSel0 and PerfEvtSel1 MSRs control the operation of the performance-monitoring counters, with one register used to set up each counter. They specify the events to be counted, how they should be counted, and the privilege levels at which counting should take place. Figure 22-65 shows the flags and fields in these MSRs.

The functions of the flags and fields in the PerfEvtSel0 and PerfEvtSel1 MSRs are as follows:

- **Event select field (bits 0 through 7)** — Selects the event logic unit to detect certain microarchitectural conditions.
- **Unit mask (UMASK) field (bits 8 through 15)** — Further qualifies the event logic unit selected in the event select field to detect a specific microarchitectural condition. For example, for some cache events, the mask is used as a MESI-protocol qualifier of cache states.

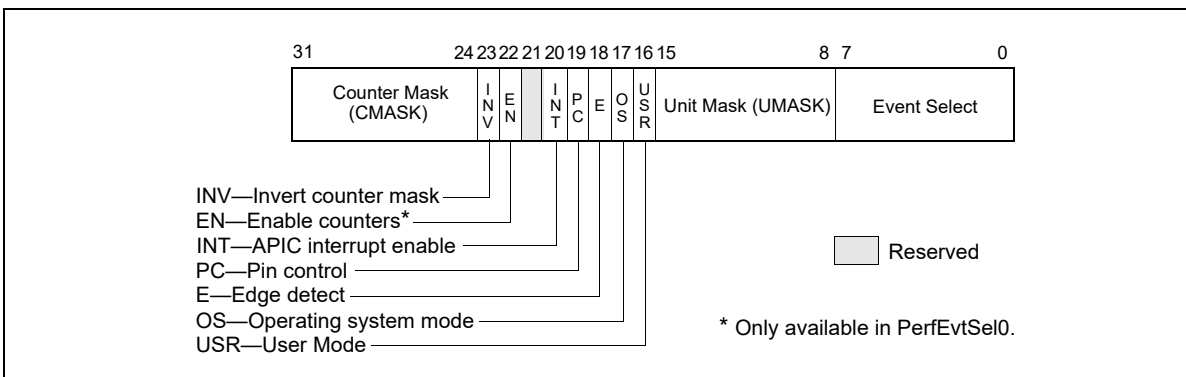


Figure 22-65. PerfEvtSel0 and PerfEvtSel1 MSRs

- **USR (user mode) flag (bit 16)** — Specifies that events are counted only when the processor is operating at privilege levels 1, 2 or 3. This flag can be used in conjunction with the OS flag.
- **OS (operating system mode) flag (bit 17)** — Specifies that events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USR flag.
- **E (edge detect) flag (bit 18)** — Enables (when set) edge detection of events. The processor counts the number of deasserted to asserted transitions of any condition that can be expressed by the other fields. The mechanism is limited in that it does not permit back-to-back assertions to be distinguished. This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).

- **PC (pin control) flag (bit 19)** — When set, the processor toggles the PMi pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PMi pins when the counter overflows. The toggling of a pin is defined as assertion of the pin for a single bus clock followed by deassertion.
- **INT (APIC interrupt enable) flag (bit 20)** — When set, the processor generates an exception through its local APIC on counter overflow.
- **EN (Enable Counters) Flag (bit 22)** — This flag is only present in the PerfEvtSel0 MSR. When set, performance counting is enabled in both performance-monitoring counters; when clear, both counters are disabled.
- **INV (invert) flag (bit 23)** — When set, inverts the counter-mask (CMASK) comparison, so that both greater than or equal to and less than comparisons can be made (0: greater than or equal; 1: less than). Note if counter-mask is programmed to zero, INV flag is ignored.
- **Counter mask (CMASK) field (bits 24 through 31)** — When nonzero, the processor compares this mask to the number of events counted during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented. This mask can be used to count events only if multiple occurrences happen per clock (for example, two or more instructions retired per clock). If the counter-mask field is 0, then the counter is incremented each cycle by the number of events that occurred that cycle.

22.6.8.2 PerfCtr0 and PerfCtr1 MSRs

The performance-counter MSRs (PerfCtr0 and PerfCtr1) contain the event or duration counts for the selected events being counted. The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

The WRMSR instruction cannot arbitrarily write to the performance-monitoring counter MSRs (PerfCtr0 and PerfCtr1). Instead, the lower-order 32 bits of each MSR may be written with any value, and the high-order 8 bits are sign-extended according to the value of bit 31. This operation allows writing both positive and negative values to the performance counters.

22.6.8.3 Starting and Stopping the Performance-Monitoring Counters

The performance-monitoring counters are started by writing valid setup information in the PerfEvtSel0 and/or PerfEvtSel1 MSRs and setting the enable counters flag in the PerfEvtSel0 MSR. If the setup is valid, the counters begin counting following the execution of a WRMSR instruction that sets the enable counter flag. The counters can be stopped by clearing the enable counters flag or by clearing all the bits in the PerfEvtSel0 and PerfEvtSel1 MSRs. Counter 1 alone can be stopped by clearing the PerfEvtSel1 MSR.

22.6.8.4 Event and Time-Stamp Monitoring Software

To use the performance-monitoring counters and time-stamp counter, the operating system needs to provide an event-monitoring device driver. This driver should include procedures for handling the following operations:

- Feature checking.
- Initialize and start counters.
- Stop counters.
- Read the event counters.
- Read the time-stamp counter.

The event monitor feature determination procedure must check whether the current processor supports the performance-monitoring counters and time-stamp counter. This procedure compares the family and model of the processor returned by the CPUID instruction with those of processors known to support performance monitoring. (The Pentium and P6 family processors support performance counters.) The procedure also checks the MSR and TSC flags returned to register EDX by the CPUID instruction to determine if the MSRs and the RDTSC instruction are supported.

The initialize and start counters procedure sets the PerfEvtSel0 and/or PerfEvtSel1 MSRs for the events to be counted and the method used to count them and initializes the counter MSRs (PerfCtr0 and PerfCtr1) to starting counts. The stop counters procedure stops the performance counters (see Section 22.6.8.3, “Starting and Stopping the Performance-Monitoring Counters”).

The read counters procedure reads the values in the PerfCtr0 and PerfCtr1 MSRs, and a read time-stamp counter procedure reads the time-stamp counter. These procedures would be provided in lieu of enabling the RDTSC and RDPMC instructions that allow application code to read the counters.

22.6.8.5 Monitoring Counter Overflow

The P6 family processors provide the option of generating a local APIC interrupt when a performance-monitoring counter overflows. This mechanism is enabled by setting the interrupt enable flag in either the PerfEvtSel0 or the PerfEvtSel1 MSR. The primary use of this option is for statistical performance sampling.

To use this option, the operating system should do the following things on the processor for which performance events are required to be monitored:

- Provide an interrupt vector for handling the counter-overflow interrupt.
- Initialize the APIC PERF local vector entry to enable handling of performance-monitor counter overflow events.
- Provide an entry in the IDT that points to a stub exception handler that returns without executing any instructions.
- Provide an event monitor driver that provides the actual interrupt handler and modifies the reserved IDT entry to point to its interrupt routine.

When interrupted by a counter overflow, the interrupt handler needs to perform the following actions:

- Save the instruction pointer (EIP register), code-segment selector, TSS segment selector, counter values and other relevant information at the time of the interrupt.
- Reset the counter to its initial setting and return from the interrupt.

An event monitor application utility or another application program can read the information collected for analysis of the performance of the profiled application.

22.6.9 Performance Monitoring (Pentium Processors)

The Pentium processor provides two 40-bit performance counters, which can be used to count events or measure duration. The counters are supported by three MSRs: the control and event select MSR (CESR) and the performance counter MSRs (CTR0 and CTR1). These can be read from and written to using the RDMSR and WRMSR instructions, respectively. They can be accessed using these instructions only when operating at privilege level 0.

Each counter has an associated external pin (PM0/BP0 and PM1/BP1), which can be used to indicate the state of the counter to external hardware.

NOTES

The CESR, CTR0, and CTR1 MSRs and the events listed for Pentium processors are model-specific for the Pentium processor.

The performance-monitoring events found at <https://perfmon-events.intel.com/> are intended to be used as guides for performance tuning. Counter values reported are not guaranteed to be accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

22.6.9.1 Control and Event Select Register (CESR)

The 32-bit control and event select MSR (CESR) controls the operation of performance-monitoring counters CTR0 and CTR1 and the associated pins (see Figure 22-66). To control each counter, the CESR register contains a 6-bit event select field (ES0 and ES1), a pin control flag (PC0 and PC1), and a 3-bit counter control field (CC0 and CC1). The functions of these fields are as follows:

- **ES0 and ES1 (event select) fields (bits 0-5, bits 16-21)** — Selects (by entering an event code in the field) up to two events to be monitored.

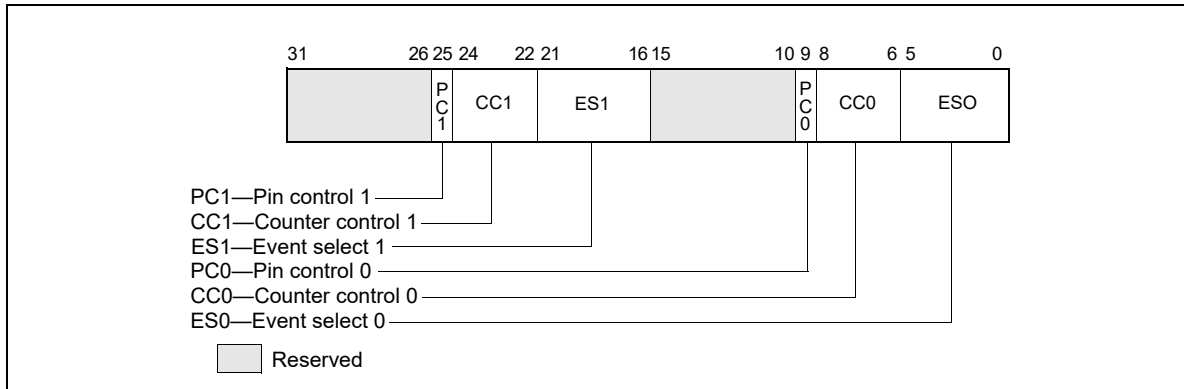


Figure 22-66. CESR MSR (Pentium Processor Only)

- **CC0 and CC1 (counter control) fields (bits 6-8, bits 22-24)** — Controls the operation of the counter. Control codes are as follows:

- 000 — Count nothing (counter disabled).
- 001 — Count the selected event while CPL is 0, 1, or 2.
- 010 — Count the selected event while CPL is 3.
- 011 — Count the selected event regardless of CPL.
- 100 — Count nothing (counter disabled).
- 101 — Count clocks (duration) while CPL is 0, 1, or 2.
- 110 — Count clocks (duration) while CPL is 3.
- 111 — Count clocks (duration) regardless of CPL.

The highest order bit selects between counting events and counting clocks (duration); the middle bit enables counting when the CPL is 3; and the low-order bit enables counting when the CPL is 0, 1, or 2.

- **PC0 and PC1 (pin control) flags (bits 9, 25)** — Selects the function of the external performance-monitoring counter pin (PM0/BP0 and PM1/BP1). Setting one of these flags to 1 causes the processor to assert its associated pin when the counter has overflowed; setting the flag to 0 causes the pin to be asserted when the counter has been incremented. These flags permit the pins to be individually programmed to indicate the overflow or incremented condition. The external signaling of the event on the pins will lag the internal event by a few clocks as the signals are latched and buffered.

While a counter need not be stopped to sample its contents, it must be stopped and cleared or preset before switching to a new event. It is not possible to set one counter separately. If only one event needs to be changed, the CESR register must be read, the appropriate bits modified, and all bits must then be written back to CESR. At reset, all bits in the CESR register are cleared.

22.6.9.2 Use of the Performance-Monitoring Pins

When performance-monitor pins PM0/BP0 and/or PM1/BP1 are configured to indicate when the performance-monitor counter has incremented and an "occurrence event" is being counted, the associated pin is asserted (high) each time the event occurs. When a "duration event" is being counted, the associated PM pin is asserted for the

entire duration of the event. When the performance-monitor pins are configured to indicate when the counter has overflowed, the associated PM pin is asserted when the counter has overflowed.

When the PM0/BP0 and/or PM1/BP1 pins are configured to signal that a counter has incremented, it should be noted that although the counters may increment by 1 or 2 in a single clock, the pins can only indicate that the event occurred. Moreover, since the internal clock frequency may be higher than the external clock frequency, a single external clock may correspond to multiple internal clocks.

A “count up to” function may be provided when the event pin is programmed to signal an overflow of the counter. Because the counters are 40 bits, a carry out of bit 39 indicates an overflow. A counter may be preset to a specific value less than $2^{40} - 1$. After the counter has been enabled and the prescribed number of events has transpired, the counter will overflow.

Approximately 5 clocks later, the overflow is indicated externally and appropriate action, such as signaling an interrupt, may then be taken.

The PM0/BP0 and PM1/BP1 pins also serve to indicate breakpoint matches during in-circuit emulation, during which time the counter increment or overflow function of these pins is not available. After RESET, the PM0/BP0 and PM1/BP1 pins are configured for performance monitoring, however a hardware debugger may reconfigure these pins to indicate breakpoint matches.

22.6.9.3 Events Counted

Events that performance-monitoring counters can be set to count and record (using CTR0 and CTR1) are divided in two categories: occurrence and duration:

- **Occurrence events** — Counts are incremented each time an event takes place. If PM0/BP0 or PM1/BP1 pins are used to indicate when a counter increments, the pins are asserted each clock counters increment. But if an event happens twice in one clock, the counter increments by 2 (the pins are asserted only once).
- **Duration events** — Counters increment the total number of clocks that the condition is true. When used to indicate when counters increment, PM0/BP0 and/or PM1/BP1 pins are asserted for the duration.

22.7 COUNTING CLOCKS

The count of cycles, also known as clockticks, forms the basis for measuring how long a program takes to execute. Clockticks are also used as part of efficiency ratios like cycles per instruction (CPI). Processor clocks may stop ticking under circumstances like the following:

- The processor is halted when there is nothing for the CPU to do. For example, the processor may halt to save power while the computer is servicing an I/O request. When Intel Hyper-Threading Technology is enabled, both logical processors must be halted for performance-monitoring counters to be powered down.
- The processor is asleep as a result of being halted or because of a power-management scheme. There are different levels of sleep. In the some deep sleep levels, the time-stamp counter stops counting.

In addition, processor core clocks may undergo transitions at different ratios relative to the processor's bus clock frequency. Some of the situations that can cause processor core clock to undergo frequency transitions include:

- TM2 transitions.
- Enhanced Intel SpeedStep Technology transitions (P-state transitions).

For Intel processors that support TM2, the processor core clocks may operate at a frequency that differs from the Processor Base frequency (as indicated by processor frequency information reported by CPUID instruction). See Section 22.7.2 for more detail.

Due to the above considerations there are several important clocks referenced in this manual:

- **Base Clock** — The frequency of this clock is the frequency of the processor when the processor is not in turbo mode, and not being throttled via Intel SpeedStep.
- **Maximum Clock** — This is the maximum frequency of the processor when turbo mode is at the highest point.
- **Bus Clock** — These clockticks increment at a fixed frequency and help coordinate the bus on some systems.

- **Core Crystal Clock** — This is a clock that runs at fixed frequency; it coordinates the clocks on all packages across the system.
- **Non-halted Clockticks** — Measures clock cycles in which the specified logical processor is not halted and is not in any power-saving state. When Intel Hyper-Threading Technology is enabled, ticks can be measured on a per-logical-processor basis. There are also performance events on dual-core processors that measure clockticks per logical processor when the processor is not halted.
- **Non-sleep Clockticks** — Measures clock cycles in which the specified physical processor is not in a sleep mode or in a power-saving state. These ticks cannot be measured on a logical-processor basis.
- **Time-stamp Counter** — See Section 20.17, “Time-Stamp Counter.”
- **Reference Clockticks** — TM2 or Enhanced Intel SpeedStep technology are two examples of processor features that can cause processor core clockticks to represent non-uniform tick intervals due to change of bus ratios. Performance events that counts clockticks of a constant reference frequency was introduced Intel Core Duo and Intel Core Solo processors. The mechanism is further enhanced on processors based on Intel Core microarchitecture.

Some processor models permit clock cycles to be measured when the physical processor is not in deep sleep (by using the time-stamp counter and the RDTSC instruction). Note that such ticks cannot be measured on a per-logical-processor basis. See Section 20.17, “Time-Stamp Counter,” for detail on processor capabilities.

The first two methods use performance counters and can be set up to cause an interrupt upon overflow (for sampling). They may also be useful where it is easier for a tool to read a performance counter than to use a time stamp counter (the timestamp counter is accessed using the RDTSC instruction).

For applications with a significant amount of I/O, there are two ratios of interest:

- **Non-halted CPI** — Non-halted clockticks/instructions retired measures the CPI for phases where the CPU was being used. This ratio can be measured on a logical-processor basis when Intel Hyper-Threading Technology is enabled.
- **Nominal CPI** — Time-stamp counter ticks/instructions retired measures the CPI over the duration of a program, including those periods when the machine halts while waiting for I/O.

22.7.1 Non-Halted Reference Clockticks

Software can use UnHalted Reference Cycles on either a general purpose performance counter using event mask 0x3C and UMASK 0x01 or on fixed function performance counter 2 to count at a constant rate. These events count at a consistent rate irrespective of P-state, TM2, or frequency transitions that may occur to the processor. The UnHalted Reference Cycles event may count differently on the general purpose event and fixed counter.

22.7.2 Cycle Counting and Opportunistic Processor Operation

As a result of the state transitions due to opportunistic processor performance operation (see Chapter 17, “Power and Thermal Management”), a logical processor or a processor core can operate at frequency different from the Processor Base frequency.

The following items are expected to hold true irrespective of when opportunistic processor operation causes state transitions:

- The time stamp counter operates at a fixed-rate frequency of the processor.
- The IA32_MPERF counter increments at a fixed frequency irrespective of any transitions caused by opportunistic processor operation.
- The IA32_FIXED_CTR2 counter increments at the same TSC frequency irrespective of any transitions caused by opportunistic processor operation.
- The Local APIC timer operation is unaffected by opportunistic processor operation.
- The TSC, IA32_MPERF, and IA32_FIXED_CTR2 operate at close to the maximum non-turbo frequency, which is equal to the product of scalable bus frequency and maximum non-turbo ratio.

22.7.3 Determining the Processor Base Frequency

For Intel processors in which the nominal core crystal clock frequency is enumerated in CPUID.15H:ECX and the core crystal clock ratio is encoded in CPUID.15H, the nominal TSC frequency can be determined by using the following equation:

$$\text{Nominal TSC frequency} = (\text{CPUID.15H:ECX}[31:0] * \text{CPUID.15H:EBX}[31:0]) \div \text{CPUID.15H:EAX}[31:0]$$

For Intel processors in which CPUID.15H:EBX[31:0] $\div$ CPUID.15H:EAX[31:0] is enumerated but CPUID.15H:ECX is not enumerated, Table 22-95 can be used to look up the nominal core crystal clock frequency.

Table 22-95. Nominal Core Crystal Clock Frequency

Processor Families/Processor Number Series ¹	Nominal Core Crystal Clock Frequency
Intel® Xeon® Scalable Processor Family with CPUID signature 06_55H.	25 MHz
6th and 7th generation Intel® Core™ processors and Intel® Xeon® W Processor Family.	24 MHz
Next Generation Intel Atom® processors based on Goldmont Microarchitecture with CPUID signature 06_5CH (does not include Intel Xeon processors).	19.2 MHz

NOTES:

- For any processor in which CPUID.15H is enumerated and MSR_PLATFORM_INFO[15:8] (which gives the scalable bus frequency) is available, a more accurate frequency can be obtained by using CPUID.15H.

22.7.3.1 For Intel® Processors Based on Sandy Bridge, Ivy Bridge, Haswell, and Broadwell Microarchitectures

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by a bus speed of 100 MHz.

22.7.3.2 For Intel® Processors Based on Nehalem Microarchitecture

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by a bus speed of 133.33 MHz.

22.7.3.3 For Intel Atom® Processors Based on Silvermont Microarchitecture (Including Intel Processors Based on Airmont Microarchitecture)

The scalable bus frequency is encoded in the bit field MSR_PLATFORM_INFO[15:8] and the nominal TSC frequency can be determined by multiplying this number by the scalable bus frequency. The scalable bus frequency is encoded in the bit field MSR_FSB_FREQ[2:0] for Intel Atom processors based on the Silvermont microarchitecture, and in bit field MSR_FSB_FREQ[3:0] for processors based on the Airmont microarchitecture; see Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

22.7.3.4 For Intel® Core™ 2 Processor Family and for Intel® Xeon® Processors Based on Intel Core Microarchitecture

For processors based on Intel Core microarchitecture, the scalable bus frequency is encoded in the bit field MSR_FSB_FREQ[2:0] at (0CDH), see Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4. The maximum resolved bus ratio can be read from the following bit field:

- If XE operation is disabled, the maximum resolved bus ratio can be read in MSR_PLATFORM_ID[12:8]. It corresponds to the Processor Base frequency.

- If XE operation is enabled, the maximum resolved bus ratio is given in MSR_PERF_STATUS[44:40], it corresponds to the maximum XE operation frequency configured by BIOS.

XE operation of an Intel 64 processor is implementation specific. XE operation can be enabled only by BIOS. If MSR_PERF_STATUS[31] is set, XE operation is enabled. The MSR_PERF_STATUS[31] field is read-only.

22.8 IA32_PERF_CAPABILITIES MSR ENUMERATION

The layout of IA32_PERF_CAPABILITIES MSR is shown in Figure 22-67; it provides enumeration of a variety of interfaces:

- IA32_PERF_CAPABILITIES.LBR_FMT[bits 5:0]: encodes the LBR format, details are described in Section 20.4.8.1.
- IA32_PERF_CAPABILITIES.PEBSTrap[6]: Trap/Fault-like indicator of PEBS recording assist; see Section 22.6.2.4.2.
- IA32_PERF_CAPABILITIES.PEBSArchRegs[7]: Indicator of PEBS assist save architectural registers; see Section 22.6.2.4.2.
- IA32_PERF_CAPABILITIES.PEBS_FMT[bits 11:8]: Specifies the encoding of the layout of PEBS records; see Section 22.6.2.4.2.
- IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[12]: Indicates IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if 1. See Section 22.8.1.
- IA32_PERF_CAPABILITIES.FULL_WRITE[13]: Indicates the processor supports IA32_A_PMCx interface for updating bits 32 and above of IA32_PMCx; see Section 22.2.8.
- IA32_PERF_CAPABILITIES.PEBS_BASELINE [bit 14]: If set, the following is true:
 - The IA32_PEBS_ENABLE MSR (address 3F1H) exists and all architecturally enumerated fixed and general-purpose counters have corresponding bits in IA32_PEBS_ENABLE that enable generation of PEBS records. The general-purpose counter bits start at bit IA32_PEBS_ENABLE[0], and the fixed counter bits start at bit IA32_PEBS_ENABLE[32].
 - The format of the PEBS record is enumerated by IA32_PERF_CAPABILITIES.PEBS_FMT; see Section 22.6.2.4.2.
 - Extended PEBS is supported. All counters support the PEBS facility, and all events (both precise and non-precise) can generate PEBS records when PEBS is enabled for that counter. Note that not all events may be available on all counters.
 - Adaptive PEBS is supported. The PEBS_DATA_CFG MSR (address 3F2H) and adaptive record enable bits (IA32_PERFEVTSELx.Adaptive_Record and IA32_FIXED_CTR_CTRL.FCx_Adaptive_Record) are supported. The definition of the PEBS_DATA_CFG MSR, including which bits are supported and how they affect the record, is enumerated by IA32_PERF_CAPABILITIES.PEBS_FMT. See Section 22.9.2.3.
 - NOTE: Software is recommended to feature PEBS Baseline when the following is true: IA32_PERF_CAPABILITIES.PEBS_BASELINE[14] && IA32_PERF_CAPABILITIES.PEBS_FMT[11:8] ≥ 4.
- IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE[15]: If set, indicates that the architecture provides built in support for TMA L1 metrics through the PERF_METRICS MSR. See Section 22.3.9.3.
- IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16]: If set on parts that enumerate support for Intel PT (CPUID.07H.00H:EBX[25]=1), setting IA32_PEBS_ENABLE.PEBS_OUTPUT to 01B will result in PEBS output being written into the Intel PT trace stream. See Section 22.5.5.2.
- IA32_PERF_CAPABILITIES.PEBS_TIMING_INFO[17]: If set, indicates that the processor supports the Timed PEBS capability. See Section 22.9.9.
- IA32_PERF_CAPABILITIES.RDPMC_METRICS_CLEAR[19]: If set, indicates that the processor supports RDPMC Metrics Clear Mode.

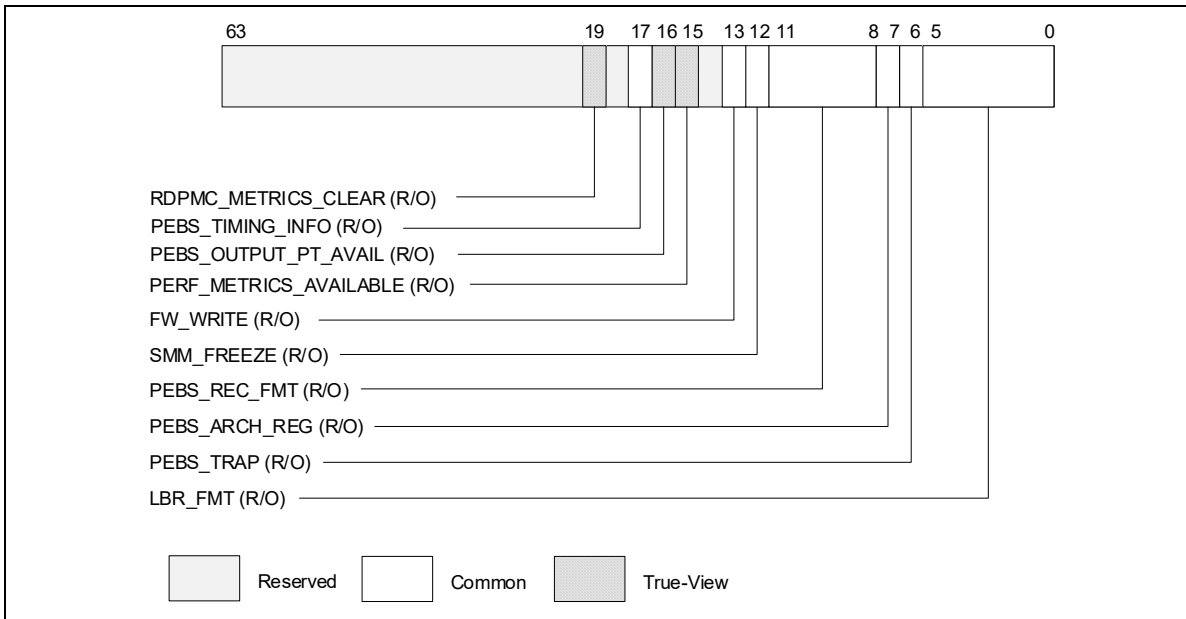


Figure 22-67. Layout of IA32_PERF_CAPABILITIES MSR

22.8.1 Filtering of SMM Handler Overhead

When performance monitoring facilities and/or branch profiling facilities (see Section 20.5, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors)”) are enabled, these facilities capture event counts, branch records and branch trace messages occurring in a logical processor. The occurrence of interrupts, instruction streams due to various interrupt handlers all contribute to the results recorded by these facilities.

If CPUID.01H:ECX.PERF_CAPABILITIES[15] is 1, the processor supports the IA32_PERF_CAPABILITIES MSR. If IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is 1, the processor supports the ability for system software using performance monitoring and/or branch profiling facilities to filter out the effects of servicing system management interrupts.

If the FREEZE_WHILE_SMM capability is enabled on a logical processor and after an SMI is delivered, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTF, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler.

The enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its servicing.

It is the responsibility of the SMM code to ensure the state of the performance monitoring and branch profiling facilities are preserved upon entry or until prior to exiting the SMM. If any of this state is modified due to actions by the SMM code, the SMM code is required to restore such state to the values present at entry to the SMM handler.

System software is allowed to set IA32_DEBUGCTL.FREEZE_WHILE_SMM[bit 14] to 1 only supported as indicated by IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] reporting 1.

22.9 PEBS FACILITY

22.9.1 Extended PEBS

The Extended PEBS feature supports Processor Event Based Sampling (PEBS) on all counters, both fixed function and general purpose; and all performance monitoring events, both precise and non-precise. PEBS can be enabled for the general purpose counters using PEBS_EN_PMCi bits of IA32_PEBS_ENABLE (i = 0, 1, ...m). PEBS can be enabled for 'i' fixed function counters using the PEBS_EN_FIXEDi bits of IA32_PEBS_ENABLE (i = 0, 1, ...n).

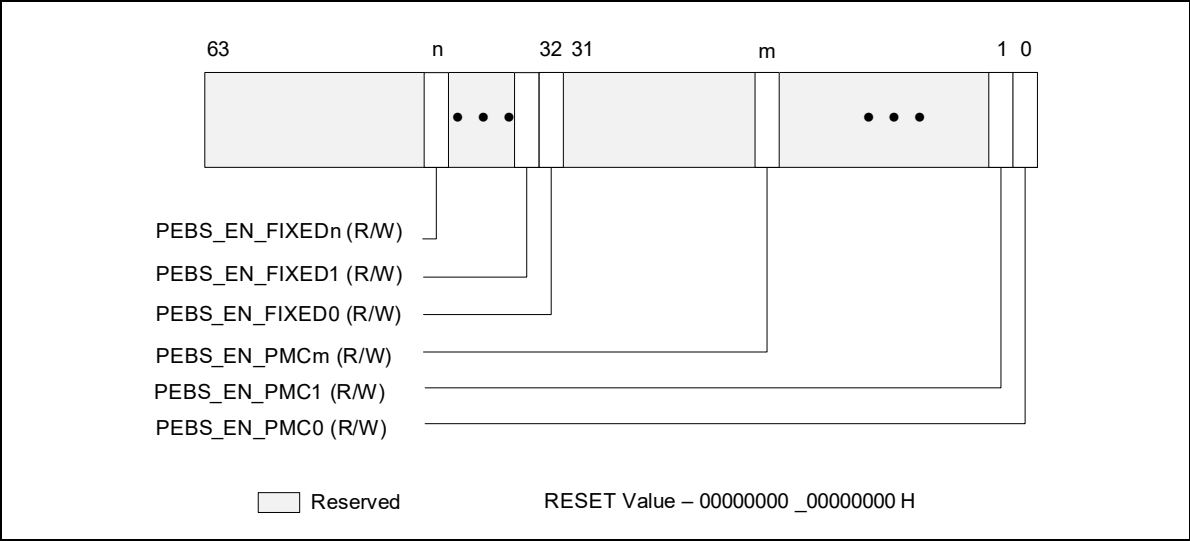


Figure 22-68. Layout of IA32_PEBS_ENABLE MSR

A PEBS record due to a precise event will be generated after an instruction that causes the event when the counter has already overflowed. A PEBS record due to a non-precise event will occur at the next opportunity after the counter has overflowed, including immediately after an overflow is set by an MSR write.

Currently, IA32_FIXED_CTR0 counts instructions retired and is a precise event. IA32_FIXED_CTR1, IA32_FIXED_CTR2 ... IA32_FIXED_CTRm count as non-precise events.

The Applicable Counter field in the Basic Info Group of the PEBS record indicates which counters caused the PEBS record to be generated. It is in the same format as the enable bits for each counter in IA32_PEBS_ENABLE. As an example, an Applicable Counter field with bits 2 and 32 set would indicate that both general purpose counter 2 and fixed function counter 0 generated the PEBS record.

To properly use PEBS for the additional counters, software will need to set up the counter reset values in PEBS portion of the DS_BUFFER_MANAGEMENT_AREA data structure that is indicated by the IA32_DS_AREA register. The layout of the DS_BUFFER_MANAGEMENT_AREA is shown in Figure 22-69. When a counter generates a PEBS records, the appropriate counter reset values will be loaded into that counter. In the above example where general purpose counter 2 and fixed function counter 0 generated the PEBS record, general purpose counter 2 would be reloaded with the value contained in PEBS GP Counter 2 Reset (offset 50H) and fixed function counter 0 would be reloaded with the value contained in PEBS Fixed Counter 0 Reset (offset 80H).

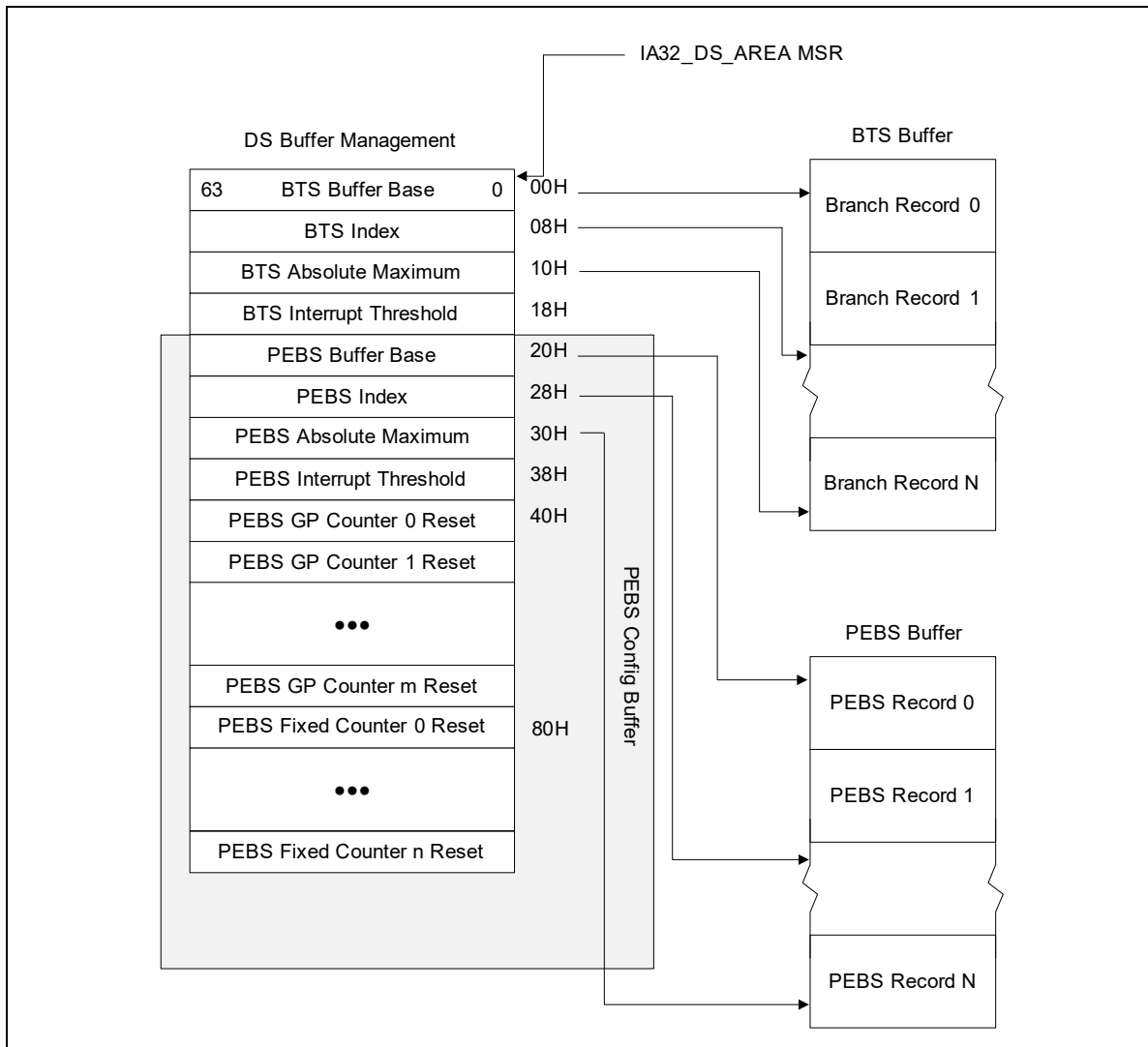


Figure 22-69. PEBS Programming Environment

Extended PEBS support debuts on Intel Atom® processors based on the Goldmont Plus microarchitecture and future Intel® Core™ processors based on the Ice Lake microarchitecture.

22.9.2 Adaptive PEBS

The PEBS facility has been enhanced to collect the following CPU state in addition to GPRs, EventingIP, TSC, and memory access related information collected by legacy PEBS:

- XMM registers
- LBR records (TO/FROM/INFO)
- Counters Snapshotting

The PEBS record is restructured where fields are grouped into Basic group, Memory group, GPR group, XMM group, LBR group, and Counters group. A new register `MSR_PEBS_DATA_CFG` provides software the capability to select data groups of interest and thus reduce the record size in memory and record generation latency. Hence, a PEBS record's size and layout vary based on the selected groups. The MSR also allows software to select LBR depth for branch data records.

By default, the PEBS record will only contain the Basic group. Optionally, each counter can be configured to generate a PEBS records with the groups specified in MSR_PEBS_DATA_CFG.

Details and examples for the Adaptive PEBS capability follow below.

22.9.2.1 Adaptive_Record Counter Control

IA32_PERFEVTSELx.Adaptive_Record[34]: If this bit is set and IA32_PEBS_ENABLE.PEBS_EN_PMCx is set for the corresponding GP counter, an overflow of PMCx results in generation of an adaptive PEBS record with state information based on the selections made in MSR_PEBS_DATA_CFG. If this bit is not set, a basic record is generated.

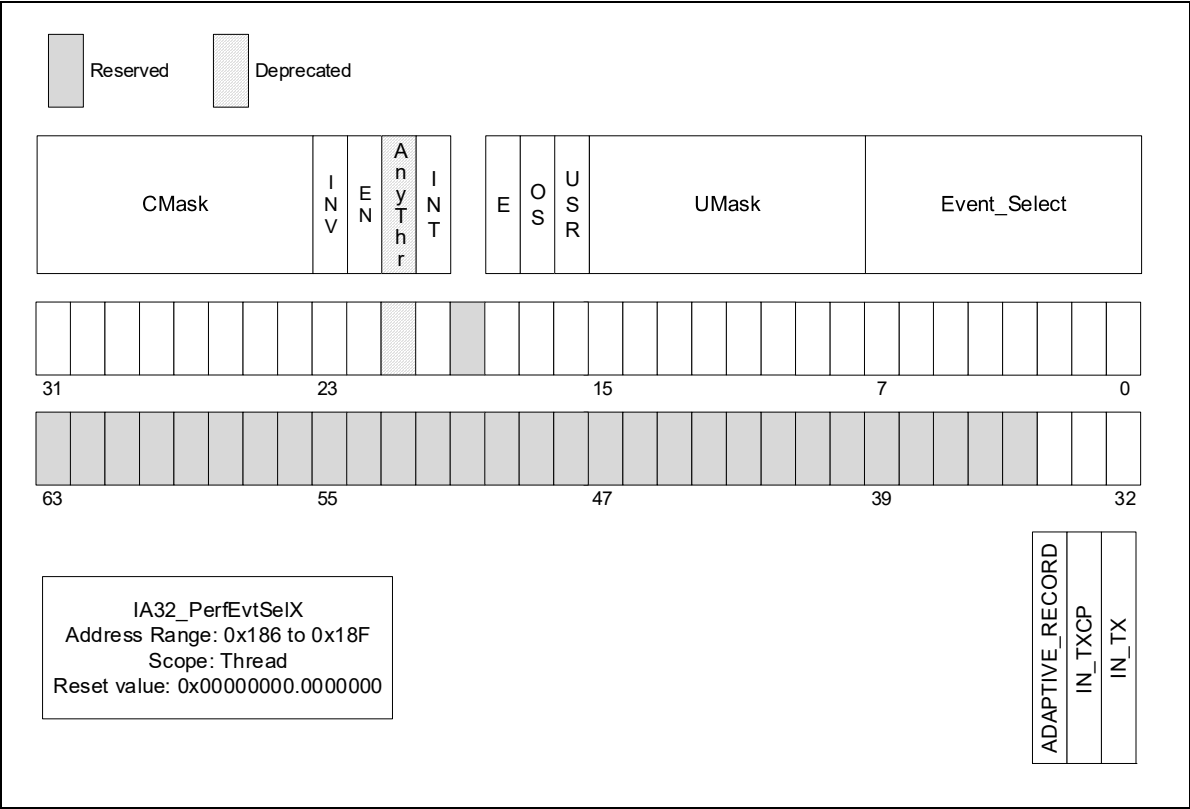


Figure 22-70. Layout of IA32_PerfEvtSelX MSR Supporting Adaptive PEBS

IA32_FIXED_CTR_CTRL.FCx_Adaptive_Record: If this bit is set and IA32_PEBS_ENABLE.PEBS_EN_FIXEDx is set for the corresponding Fixed counter, an overflow of FixedCtrx results in generation of an adaptive PEBS record with state information based on the selections made in MSR_PEBS_DATA_CFG. If this bit is not set, a basic record is generated.

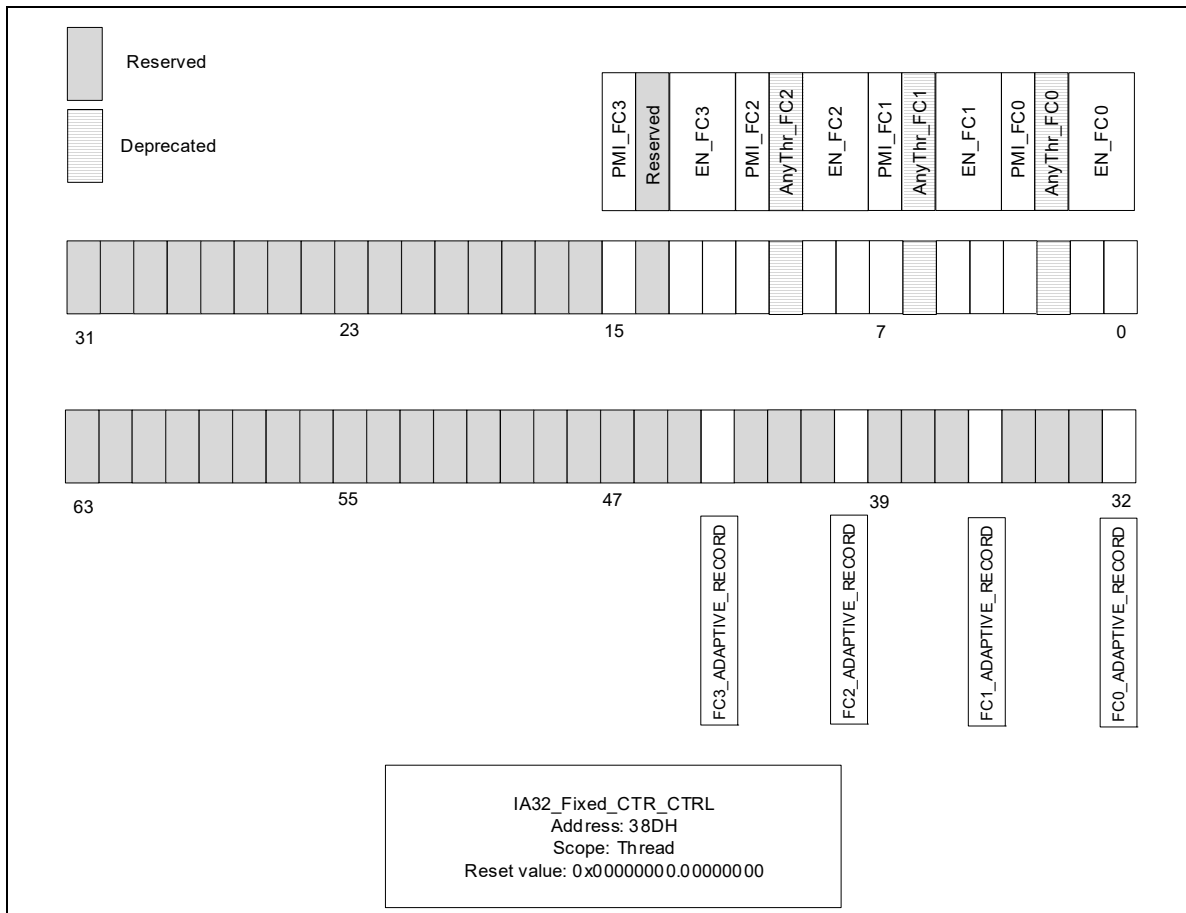


Figure 22-71. Layout of IA32_FIXED_CTR_CTRL MSR Supporting Adaptive PEBS

22.9.2.2 PEBS Record Format

The data fields in the PEBS record are aggregated into five groups which are described in the sub-sections below. Processors that support Adaptive PEBS implement a new MSR called MSR_PEBS_DATA_CFG which allows software to select the data groups to be captured. The data groups are not placed at fixed locations in the PEBS record, but are positioned immediately after one another, thus making the record format/size variable based on the groups selected.

22.9.2.2.1 Basic Info

The Basic group contains essential information for software to parse a record along with several critical fields. It is always collected.

Table 22-96. Basic Info Group

Field Name	Bit Width	Description
Record Format	[47:0]	This field indicates which data groups are included in the record. The field is zero if none of the counters that triggered the current PEBS record have their Adaptive_Record bit set. Otherwise it contains the value of MSR_PEBS_DATA_CFG.
	[63:48]	This field provides the size of the current record in bytes. Selected groups are packed back-to-back in the record without gaps or padding for unselected groups.

Table 22-96. Basic Info Group (Contd.)

Instruction Pointer	[63:0]	This field reports the Eventing Instruction Pointer (EventingIP) of the retired instruction that triggered the PEBS record generation. Note that this field is different than R/EIP which records the instruction pointer of the next instruction to be executed after record generation. The legacy R/EIP field has been removed.
Applicable Counters	[63:0]	The Applicable Counters field indicates which counters triggered the generation of the PEBS record, linking the record to specific events. This allows software to correlate the PEBS record entry properly with the instruction that caused the event, even when multiple counters are configured to generate PEBS records and multiple bits are set in the field.
TSC	[63:0]	This field provides the time stamp counter value when the PEBS record was generated.

22.9.2.2.2 Memory Access Info

This group contains the legacy PEBS memory-related fields; see Section 22.3.1.1.2.

Table 22-97. Memory Access Info Group

Field Name	Bit Width	Description
Memory Access Address	[63:0]	This field contains the linear address of the source of the load, or linear address of the destination (target) of the store. This value is written as a 64-bit address in canonical form.
Memory Auxiliary Info	[63:0]	When a MEM_TRANS_RETIRE.* event is configured in a General Purpose counter, this field contains an encoded value indicating the memory hierarchy source which satisfied the load. These encodings are detailed in Table 22-5 and Table 22-14. If the PEBS assist was triggered for a store uop, this field will contain information indicating the status of the store, as detailed in Table 22-15.
Memory Access Latency ¹	[63:0]	When a MEM_TRANS_RETIRE.* event is configured in a General Purpose counter, this field contains the latency to service the load in core clock cycles.
TSX Auxiliary Info	[31:0]	This field contains the number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
	[63:32]	This field contains the abort details. Refer to Section 22.3.6.5.1.

NOTES:

1. In certain conditions, high latencies in fields under “Memory Access Latency” may be observed even when the Data Src of the “Memory Auxiliary Info” field indicates a close source.

Beginning with 12th generation Intel Core processors, the memory access information group has been updated. New fields added are shaded gray in Table 22-98.

Table 22-98. Updated Memory Access Info Group

Field Name	Sub-field Name	Bits	Description
Access Address (offset 0H)	DLA	[63:0]	This field reports the data linear address (DLA) of the memory access in canonical form. A zero value indicates the processor could not retrieve the address of the particular access.
Access Info (offset 8H)	Data Src	[3:0]	An encoded value indicating the memory hierarchy source which satisfied the access. These encodings are detailed in Table 22-5. A zero value indicates the processor could not retrieve the data source of the particular access.
	STLB-miss	[4]	A value of 1 indicates the access has missed the Second-level TLB (STLB).
	Is-Lock	[5]	A value of 1 indicates the access was part of a locked (atomic) memory transaction.
	Data-Blk	[6]	A value of 1 indicates the load was blocked since its data could not be forwarded from a preceding store.
	Address-Blk	[7]	A value of 1 indicates the load was blocked due to potential address conflict with a preceding store.
Access Latency (offset 10H)	Instruction Latency	[15:0]	Measured instruction latency in core cycles. For loads, the latency starts by the dispatch of the load operation for execution and lasts until completion of the instruction it belongs to. This field includes the entire latency including time for data-dependency resolution or TLB lookups.
	Cache Latency	[47:32]	Measured cache access latency in core cycles. For loads, the latency starts by the actual cache access until the data is returned by the memory subsystem. For stores, the latency starts when the demand write accesses the L1 data-cache and lasts until the cacheline write is completed in the memory subsystem. This field does not include non-data-cache latency such as memory ordering checks or TLB lookups.
TSX (offset 18H)	Transaction Latency	[31:0]	This field contains the number of cycles in the last TSX region, regardless of whether that region had aborted or committed.
	Abort Info	[63:32]	This field contains the abort details. Refer to Section 22.3.6.5.1.

To determine which fields are supported for certain performance monitoring events, consult the Memory Info attribute in the event lists at <https://github.com/intel/perfmon>.

NOTE

There may be additional block reasons, even if Data-Blk and Address-Blk are both clear, e.g., non-optimal instruction latency.

On P-core, the new Data-Blk and Address-Blk bits require the event LD_BLOCKS.STORE_FORWARD (r8203) to be configured in a programmable counter.

22.9.2.2.3 GPRs

This group is captured when the GPR bit is enabled in MSR_PEBS_DATA_CFG. GPRs are always 64 bits wide. If they are selected for non 64-bit mode, the upper 32-bit of the legacy RAX - RDI and all contents of R8-15 GPRs will be filled with 0s. In 64bit mode, the full 64 bit value of each register is written.

The order differs from legacy. The table below shows the order of the GPRs in Ice Lake microarchitecture.

Table 22-99. GPRs in Ice Lake Microarchitecture

Field Name	Bit Width
RFLAGS	[63:0]
RIP	[63:0]
RAX	[63:0]
RCX*	[63:0]
RDX*	[63:0]
RBX*	[63:0]
RSP*	[63:0]
RBP*	[63:0]
RSI*	[63:0]
RDI*	[63:0]
R8	[63:0]
...	...
R15	[63:0]

The machine state reported in the PEBS record is the committed machine state immediately after the instruction that triggers PEBS completes.

For instance, consider the following instruction sequence:

MOV eax, [eax]; triggers PEBS record generation

NOP

If the mov instruction triggers PEBS record generation, the EventingIP field in the PEBS record will report the address of the mov, and the value of EAX in the PEBS record will show the value read from memory, not the target address of the read operation. And the value of RIP will contain the linear address of the nop.

22.9.2.2.4 XMMs

This group is captured when the XMM bit is enabled in MSR_PEBS_DATA_CFG and SSE is enabled. If SSE is not enabled, the fields will contain zeroes. XMM8-XMM15 will also contain zeroes if not in 64-bit mode.

Table 22-100. XMMs

Field Name	Bit Width
XMM0	[127:0]
...	...
XMM15	[127:0]

22.9.2.2.5 LBRs

To capture LBR data in the PEBS record, the LBR bit in MSR_PEBS_DATA_CFG must be enabled. The number of LBR entries included in the record can be configured in the LBR_entries field of MSR_PEBS_DATA_CFG.

Table 22-101. LBRs

Field Name	Bit Width	Description
LBR[.].FROM	[63:0]	Branch from address.
LBR[.].TO	[63:0]	Branch to address.
LBR[.].INFO	[63:0]	Other LBR information, like timing. This field is described in more detail in Section 20.12.1, “MSR_LBR_INFO_x MSR.”

LBR entries are recorded into the record starting at LBR[TOS] and proceeding to LBR[TOS-1] and following. Note that LBR index is modulo the number of LBRs supporting on the processor.

22.9.2.3 MSR_PEBS_DATA_CFG

Bits in MSR_PEBS_DATA_CFG can be set to include data field blocks/groups into adaptive records. The Basic Info group is always included in the record. Additionally, the number of LBR entries included in the record is configurable.

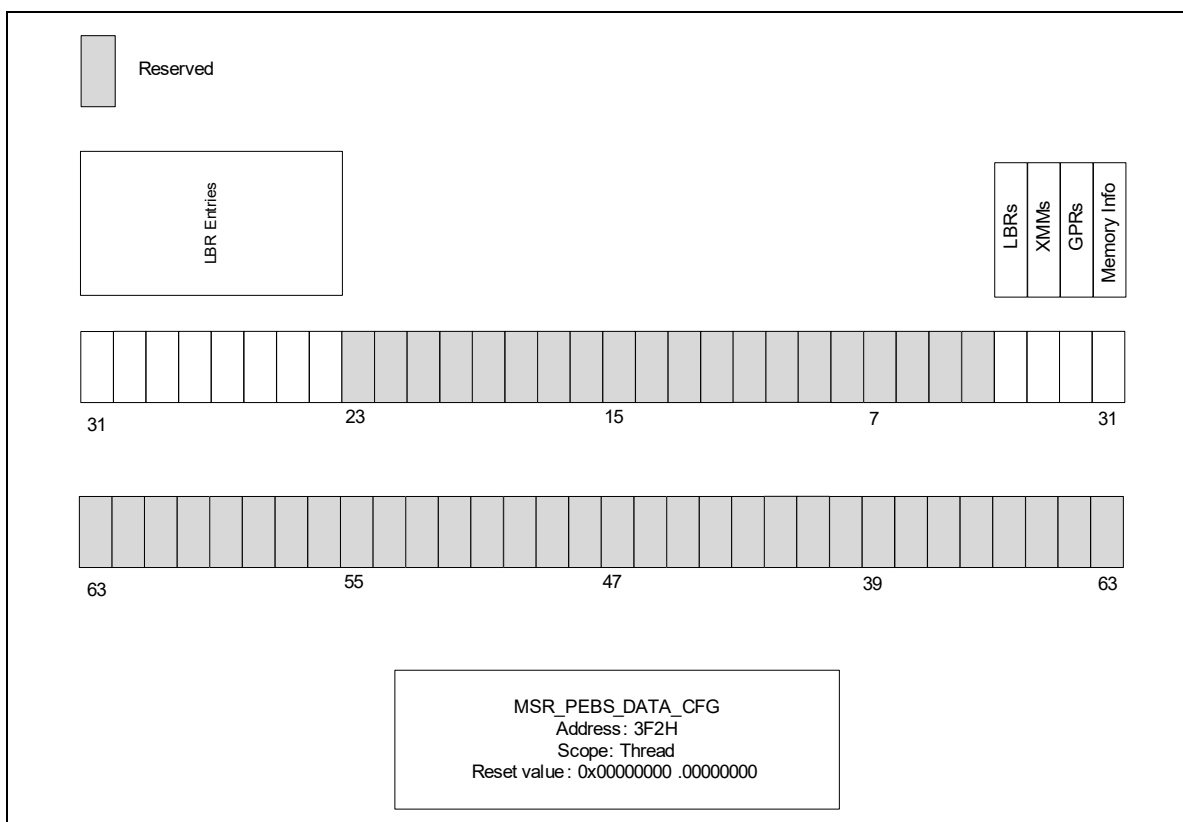


Figure 22-72. Legacy MSR_PEBS_DATA_CFG

Beginning with the Intel Series 2 Core Ultra processor, which counters are included in the Counters group is configurable. See Figure 22-73.

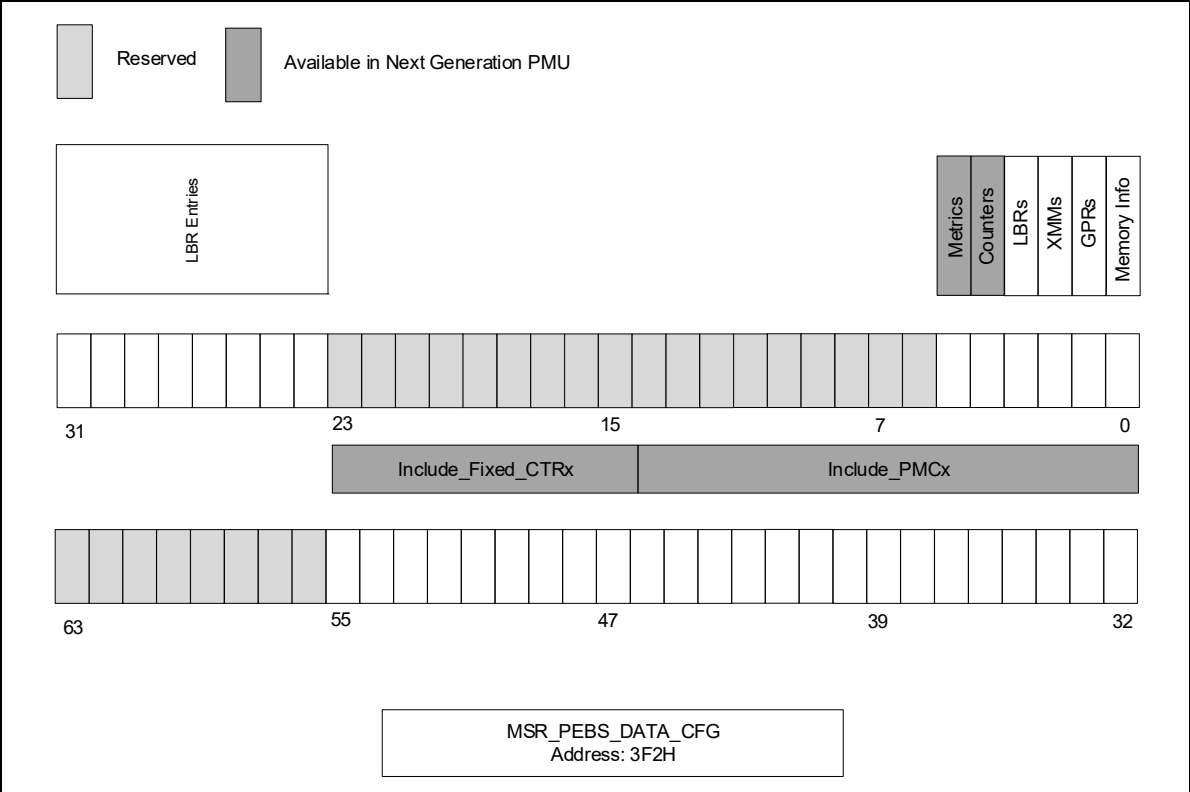


Figure 22-73. MSR_PEBS_DATA_CFG in PEBS_FMT=6

Table 22-102. MSR_PEBS_CFG Programming¹

Bit Name	Bit Index	Access	Description	Availability
Memory Info	0	R/W	Setting this bit will capture memory information such as the linear address, data source and latency of the memory access in the PEBS record.	PEBS_FMT=4 and later
GPRs	1	R/W	Setting this bit will capture the contents of the General Purpose registers in the PEBS record.	PEBS_FMT=4 and later
XMMs	2	R/W	Setting this bit will capture the contents of the XMM registers in the PEBS record.	PEBS_FMT=4 and later
LBRs	3	R/W	Setting this bit will capture LBR TO, FROM, and INFO in the PEBS record.	PEBS_FMT=4 and later
Counters	4	R/W	Setting this bit will allow recording of the IA32_PMCx MSRs and the IA32_FIXED_CTRx counters. The Include_PMCx and Include_Fixed_CTRx bits are also set.	PEBS_FMT=6 ²
Metrics	5	R/W	Setting this bit will allow recording and clearing of the MSR_PERF_METRICS register (when the Include_Fixed_CTR3 bit is also set).	PEBS_FMT=6 ² && PERF_METRICS_AVAILABLE = 1
Reserved ³	23:6	NA	Reserved.	

Table 22-102. MSR_PEBS_CFG Programming¹ (Contd.)

LBR Entries	31:24	R/W	Set the field to the desired number of entries minus 1. For example, if the LBR_entries field is 0, a single entry will be included in the record. To include 32 LBR entries, set the LBR_entries field to 31 (0x1F). To ensure all PEBS records are 16-byte aligned, it is recommended to select an even number of LBR entries (programmed into LBR_entries as an odd number).	PEBS_FMT=4 and later
Include_PMCx	47:32	R/W	A bit mask of the general-purpose counters that are allowed to be captured into the PEBS record. Note that only bits that match reporting of CPUID.23H.01H:EAX are writable.	PEBS_FMT=6 ²
Include_FIXED_CTRx	55:48	R/W	A bit mask of the fixed-function counters that are allowed to be captured into the PEBS record. Note that only bits that match reporting of CPUID.23H.01H:EBX are writable.	PEBS_FMT=6 ²
Reserved	63:56	NA	Reserved.	

NOTES:

1. A write to the MSR will be ignored when IA32_MISC_ENABLE.PERFMON_AVAILABLE is zero (default).
2. These fields are available starting with the IA32_PERF_CAPABILITIES.PEBS_FMT of 6 in addition to a subset of processors with a CPUID signature value of DisplayFamily_DisplayModel 06_C5H or 06_C6H (though they report IA32_PERF_CAPABILITIES.PEBS_FMT as 5).
3. Writing to the reserved bits will cause a GP fault.

22.9.2.3.1 Counters and Metrics Group

To capture the counters group, either the COUNTERS bit or the METRICS bit must be enabled in MSR_PEBS_DATA_CFG. The group allows recording of the IA32_PMCx MSRs, IA32_FIXED_CTRx MSRs, and the Performance Metrics.

The counters group first captures a 128-bit header with the bit vector of the counters that are captured later. The format of the counters header and the payload is shown in Table 22-103.

The group is available starting with IA32_PERF_CAPABILITIES.PEBS_FMT of 6. Additionally, the group is available in a subset of processors with a CPUID signature value of DisplayFamily_DisplayModel 06_C5H or 06_C6H (though they report IA32_PERF_CAPABILITIES.PEBS_FMT as 5).

Table 22-103. Counters Group

Field Name	Sub-Field Name	Bit Width	Description
Counters Group Header	PMC BitVector	[31:0]	Bit vector of IA32_PMCx MSRs. IA32_PMCx is recorded if bit x is set.
	FIXED_CTR BitVector	[31:0]	Bit vector of IA32_FIXED_CTRx MSRs. IA32_FIXED_CTRx is recorded if bit x is set.
	Metrics BitVector	[31:0]	Bit vector of the performance metrics counters.
	Reserved	[31:0]	Reserved.

Table 22-103. Counters Group (Contd.)

Field Name	Sub-Field Name	Bit Width	Description
Counters/Metrics Values	PMCx	[63:0]	PMCx will be captured if PMC BitVector x is set.
	...		
	FIXED_CTRx	[63:0]	FIXED_CTRx will be captured if FIXED_CTRx BitVector x is set.
	...		
	Metrics Base	[63:0]	The performance metrics base, mapped to IA32_FIXED_CTR3, if Metrics BitVector bit 0 is set.
	Metrics Data	[63:0]	MSR_PERF_METRICS, if Metrics BitVector bit 1 is set.

IA32_PMCx will be captured if both Counters and MSR_PEBS_DATA_CFG bit 32 + x are set. In this case, the PMC BitVector field bit x will be set too.

IA32_FIXED_CTRx will be captured if both Counters and MSR_PEBS_DATA_CFG bit 48 + x are set. In this case, the FIXED_CTR BitVector field bit x will be set too.

The performance metrics will be recorded if both Metrics and MSR_PEBS_DATA_CFG bit 51 (the bit used for IA32_FIXED_CTR3) are set. The Metrics record will have two 64-bit fields, MSR_PERF_METRICS and the PERF_METRICS_BASE that is derived from IA32_FIXED_CTR3. In this case, the Metrics BitVector will be 3. Note that MSR_PERF_METRICS and the IA32_FIXED_CTR3 MSR will be cleared after they are recorded.

Size of the group can be calculated in bytes by: 16 + popcount(BitVectors[127:0]) * 8.

22.9.2.4 PEBS Record Examples

The following example shows the layout of the PEBS record when all data groups are selected (all valid bits in MSR_PEBS_DATA_CFG are set) and maximum number of LBRs are selected. There are no gaps in the PEBS record when a subset of the groups are selected, thus keeping the layout compact. Implementations that do not support some features will have to pad zeroes in the corresponding fields.

Table 22-104. PEBS Record Example 1

Offset	Group Name	Field Name	Legacy Name (If Different)
0x0	Basic Info	Record Format	New
		Record Size	New
0x8		Instruction Pointer	EventingRIP
0x10		Applicable Counters	
0x18		TSC	
0x20	Memory Info	Memory Access Address	DLA
0x28		Memory Auxiliary Info	DATA_SRC
0x30		Memory Access Latency	Load Latency
0x38		TSX Auxiliary Info	HLE Information

Table 22-104. PEBS Record Example 1 (Contd.)

0x40	GPRs	RFLAGS	
0x48		RIP	
0x50		RAX	
...		...	
0x88		RDI	
0x90		R8	
...		...	
0xC8		R15	
0xD0	XMMs	XMM0	New
...		...	
0x1C0		XMM15	
0x1D0	LBRs	LBR[TOS].FROM	New
0x1D8		LBR[TOS].TO	
0x1E0		LBR[TOS].INFO	
...		...	
0x4B8		LBR[TOS + 1].FROM	
0x4C0		LBR[TOS + 1].TO	
0x4C8		LBR[TOS + 1].INFO	

The following example shows the layout of the PEBS record when Basic, GPR, and LBR group with 3 LBR entries are selected.

Table 22-105. PEBS Record Example 2

Offset	Group Name	Field Name	Legacy Name (If Different)
0x0	Basic Info	Record Format	New
		Record Size	New
0x8		Instruction Pointer	EventingRIP
0x10		Applicable Counters	
0x18		TSC	
0x20	GPRs	RFLAGS	
0x28		RIP	
0x30		RAX	
...		...	
0x68		RDI	
0x70		R8	
...		...	
0xA8		R15	
0xB0	LBRs	LBR[TOS].FROM	New
0xB8		LBR[TOS].TO	
0xC0		LBR[TOS].INFO	
...		...	
0xE0		LBR[TOS + 1].FROM	
0xE8		LBR[TOS + 1].TO	
0xF0		LBR[TOS + 1].INFO	

22.9.3 Precise Distribution of Instructions Retired (PDIR) Facility

Precise Distribution of Instructions Retired Facility is available via PEBS on some microarchitectures. Refer to Section 22.3.4.4.4. Counters that support PDIR also vary. See the processor specific sections for availability.

22.9.4 Reduced Skid PEBS

For precise events, upon triggering a PEBS assist, there will be a finite delay between the time the counter overflows and when the microcode starts to carry out its data collection obligations. The Reduced Skid mechanism mitigates the “skid” problem by providing an early indication of when the counter is about to overflow, allowing the machine to more precisely trap on the instruction that actually caused the counter overflow thus greatly reducing skid.

This mechanism is a superset of the PDIR mechanism available in the Sandy Bridge microarchitecture. See Section 22.3.4.4.4

In the Goldmont microarchitecture, the mechanism applies to all precise events including, INST_RETIRE, except for UOPS_RETIRE. However, the Reduced Skid mechanism is disabled for any counter when the INV, ANY, E, or CMASK fields are set.

With Reduced Skid PEBS, the skid is precisely one event occurrence. Hence if counting INST_RETIRE, PEBS will indicate the instruction that follows that which caused the counter to overflow.

For the Reduced Skid mechanism to operate correctly, the performance monitoring counters should not be reconfigured or modified when they are running with PEBS enabled. The counters need to be disabled (e.g., via IA32_PERF_GLOBAL_CTRL MSR) before changes to the configuration (e.g., what event is specified in IA32_PERFEVTSELx or whether PEBS is enabled for that counter via IA32_PEBS_ENABLE) or counter value (MSR write to IA32_PMCx and IA32_A_PMCx).

22.9.5 EPT-Friendly PEBS

The 3rd generation Intel Xeon Scalable Family of processors based on Ice Lake microarchitecture (and later processors) and the 12th generation Intel Core processor (and later processors) support VMX guest use of PEBS when the DS Area (including the PEBS Buffer and DS Management Area) is allocated from a paged pool of EPT pages. In such a configuration PEBS DS Area accesses may result in VM exits (e.g., EPT violations due to “lazy” EPT page-table entry propagation), and in such cases the PEBS record will not be lost but instead will “skid” to after the subsequent VM Entry back to the guest. For precise events the guest will observe that the record skid by one event occurrence, while for non-precise events the record will skid by one instruction.

22.9.6 PDist: Precise Distribution

PDist eliminates any skid or shadowing effects from PEBS. With PDist, the PEBS record will be generated precisely upon completion of the instruction or operation that causes the counter to overflow (there is no “wait for next occurrence” by default).

PDist is supported by selected counters, and is only supported when those counters are programmed to count select precise events¹. The legacy PEBS behavior applies to counters that do not support PDist, unless specified otherwise. PDist requires that the INV, ANY, E, EQ, and CMASK fields are cleared. Which counters support PDist, and which events are supported for PDist, is model-specific. Further, the counter reload value must not be less than 256 for PDist to operate.

For the PDist mechanism to operate correctly, the performance monitoring counters should not be reconfigured or modified when they are running with PEBS enabled. The counters need to be disabled (e.g., via IA32_PERF_GLOBAL_CTRL MSR) before changes to the configuration (e.g., what event is specified in IA32_PERFEVTSELx or IA32_FIXED_CTR_CTRL or whether PEBS is enabled for that counter via IA32_PEBS_ENABLE) or counter value (MSR write to IA32_PMCx and IA32_A_PMCx or IA32_FIXED_CTRx).

22.9.7 Load Latency Facility

The load latency facility provides software a means to characterize the latencies of memory load operations to different levels of cache/memory hierarchy. This facility requires a processor supporting the enhanced PEBS record format in the PEBS buffer.

Beginning with 12th generation Intel Core processors, the load latency facility supports all fields in Table 22-98, “Updated Memory Access Info Group,” in addition to the Memory Access Address field:

- The **Instruction Latency** field measures the load latency from the load's first dispatch until final data writeback from the memory subsystem. The latency is reported for retired demand load operations and in core cycles (it accounts for re-dispatches and data dependencies).
- The **Cache Latency** field measures the subset of cache access latency in core cycles. It starts from the actual cache access until the data is returned by the memory subsystem. The latency is reported for retired demand load operations in core cycles (it does not account for memory ordering blocks).
- The **Data Source** field is an encoded value indicates the origin of the data obtained by the load instruction. The encoding is shown in Table 22-106. “Data Source Encoding for Memory Accesses (Ice Lake and Later Microarchitectures)”. In the descriptions, local memory refers to system memory physically attached to a

1. To determine whether an event is precise or supports PDist, consult the relevant attribute in the event lists at <https://github.com/intel/perfmon>.

processor package, and remote memory refers to system memory or cache physically attached to another processor package (in a server product).

- Through the **Access Info** field, load latency features binary indications on certain blocks that the load operation may have encountered. Refer to STLB-miss, Is-Lock, Data-Blk and Address-Blk fields in Table 22-98.

NOTE

For loads triggered by software prefetch instructions, the cache related fields including Data Source and Cache Latency, report values as if the load was an L1 cache hit (the prefetch completes without waiting for data return, for performance reasons).

Table 22-106. Data Source Encoding for Memory Accesses (Ice Lake and Later Microarchitectures)

Encoding [3:0]	Description
00H	Unknown Data Source (the processor could not retrieve the origin of this request).
01H	L1 HIT. This request was satisfied by the L1 data cache. (Minimal latency core cache hit.)
02H	FB HIT. This request was merged into an outstanding cache miss to same cache-line address.
03H	L2 HIT. This request was satisfied by the L2 cache.
04H	L3 HIT. This request was satisfied by the L3 cache with no coherency actions performed (snooping).
05H	XCORE MISS. This request was satisfied by the L3 cache but involved a coherency check in some sibling core(s).
06H	XCORE HIT. This request was satisfied by the L3 cache but involved a coherency check that hit a non-modified copy in a sibling core.
07H	XCORE FWD. This request was satisfied by a sibling core where either a modified (cross-core HITM) or a non-modified (cross-core FWD) cache-line copy was found.
08H	Local Far Memory. This request has missed the L3 cache and was serviced by local far memory.
09H	Remote Far Memory. This request has missed the L3 cache and was serviced by remote far memory.
0AH	Local Near Memory. This request has missed the L3 cache and was serviced by local near memory.
0BH	Remote Near Memory. This request has missed the L3 cache and was serviced by remote near memory.
0CH	Remote FWD. This request has missed the L3 cache and a non-modified cache-line copy was forwarded from a remote cache.
0DH	Remote HITM. This request has missed the L3 cache and a modified cache-line was forwarded from a remote cache.
0EH	I/O. Request of input/output operation.
0FH	UC. The request was to uncacheable memory.

Table 22-107. Data Source Encoding for Memory Accesses (Lion Cove and Next Generation Microarchitectures)

Encoding [4:0]	Description
00H	Unknown Data Source (the processor could not retrieve the origin of this request).
01H or 02H	L1 HIT. This request was satisfied by the L1 data cache. (Minimal latency core cache hit.)
03H	FB merge. L1 mishandling buffer.
05H	L2 HIT. This request was satisfied by the L2 cache.
06H	XQ merge. L2 mishandling buffer.
08H	L3 HIT. This request was satisfied by the L3 cache.
0CH	L3 Hit, x-core forward.
0DH	L3 Hit, x-core modified.
0FH	L3 Miss, x-core modified.

Table 22-107. Data Source Encoding for Memory Accesses (Lion Cove and Next Generation Microarchitectures)

Encoding [4:0]	Description
10H	L3 Miss, MSC Hit (memory-side cache).
11H	L3 Miss, memory.

To use this feature, software must complete the following steps:

- Complete the PEBS configuration steps.
- Set the Memory Info bit in the PEBS_DATA_CFG MSR.
- One of the relevant IA32_PERFEVTSELx MSRs is programmed to specify the event unit MEM_TRANS_RETIRED.LOAD_LATENCY (IA32_PerfEvtSelX[15:0] = 1CDH). The corresponding counter, IA32_PMCx, will accumulate event counts for architecturally visible loads which exceed the programmed latency threshold specified separately in an MSR. Stores are ignored when this event is programmed. The CMASK or INV fields of the IA32_PerfEvtSelX register used for counting load latency must be 0. Writing other values will result in undefined behavior.
- The MSR_PEBS_LD_LAT_THRESHOLD MSR is programmed with the desired latency threshold in core clock cycles. Loads with instruction latency greater than this value are eligible for counting and PEBS data reporting. The minimum value that may be programmed in this register is 1.
- The PEBS enable bit in the IA32_PEBS_ENABLE register is set for the corresponding IA32_PMCx counter register.

Refer to Section 22.3.4.4.2 for further implementation details of Load Latency.

22.9.8 Store Latency Facility

Store latency support is available on the 12th generation Intel Core processor. Store latency is a PEBS extension that provides a means to profile store memory accesses in the system. It complements the load latency facility.

Store latency leverages the PEBS facility where it can provide additional information about sampled stores. The additional information includes the data address, memory auxiliary information, and the cache latency of the store access. Normal stores (those preceded with a read-for-ownership) as well as streaming stores are supported by the store latency facility.

Memory store operations typically do not limit performance since they update the memory with no operation that directly depends on them. Thus, data out of this facility should be carefully used once stores are suspected as a performance limiter; for example, once the TMA node of Backend_Bound.Memory_Bound.Store_Bound is flagged¹.

To enable the store latency facility, software must complete the following steps:

- Complete the PEBS configuration steps.
- Set the Memory Info bit in the PEBS_DATA_CFG MSR.
- Program the MEM_TRANS_RETIRED.STORE_SAMPLE event on general-purpose performance-monitoring counter 0 (IA32_PERFEVTSEL0[15:0] = 2CDH).
- Setup the PEBS buffer to hold at least two records, setting both 'PEBS Absolute Maximum' and 'PEBS Interrupt Threshold', should any other counter be used by PEBS (that is whenever IA32_PEBS_ENABLE[x] ≠ 0 for x ≠ 0).
- Set IA32_PEBS_ENABLE[0].

The store latency information is written into a PEBS record as shown in Table 22-49. "MSR_OFFCORE_RSP_x Supplier Info Field Definition (Processors Based on Ice Lake Microarchitecture)".

The store latency relies on the PEBS facility, so the PEBS configuration must be completed first. Unlike load latency, there is no option to filter on a subset of stores that exceed a certain threshold.

1. For more details about the method, refer to Section B.1, "Top-Down Analysis Method" of the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

22.9.9 Timed Processor Event Based Sampling

Timed Processor Event Based Sampling (Timed PEBS) enables recording of time in every PEBS record. It extends all PEBS records with timing information in a new “Retire Latency” field that is placed in the Basic Info group of the PEBS record as shown in Table 22-108.

Table 22-108. PEBS Basic Info Group

Offset	Field Name	Bits
0x0	Record Format	[31:0]
	Retire Latency	[47:32]
	Record Size	[63:48]
0x08	Instruction Pointer	[63:0]
0x10	Applicable Counters	[63:0]
0x18	TSC	[63:0]

The Retire Latency field reports the number of Unhalted Core Cycles between the retirement of the current instruction (as indicated by the Instruction Pointer field of the PEBS record) and the retirement of the prior instruction. All ones are reported when the number exceeds 16 bits.

Processors that support this enhancement set a new bit: IA32_PERF_CAPABILITIES.PEBS_TIMING_INFO[bit 17].

NOTE

Timed PEBS is not supported when PEBS is programmed on fixed-function counter 0. The Retire Latency field of such record is undefined.

22.9.10 Counters Snapshotting

Counters Snapshotting extends Adaptive PEBS with the PEBS Counters and Metrics group. This extension enables software to capture general-purpose counters, fixed-function counters, and performance metrics in the PEBS record. For additional details, see Section 22.9.2.3.1, “Counters and Metrics Group.”

22.10 ARCHITECTURAL PEBS

Processor Event-Based Sampling (PEBS) is a non-destructive profiling facility that enables capturing the CPU state, e.g., the Instruction Pointer register, memory access address, etc., when the counter of a Performance Monitoring (PerfMon) event, such as a cache miss or a retired instruction, overflows. Architectural PEBS creates a permanent, enumerated, and stable event-based sampling architecture based on the positive evolution of legacy PEBS.

A counter can select, independently from other counters, any of the following state groups to be included on its samples:

- Basic group (Instruction Pointer, time-stamp)
- Memory Auxiliary group
- General-Purpose Registers group
- XSAVE-Enabled Registers (XER) group
- Last Branch Records (LBRs) group
- Performance-monitoring Counters group

Additionally, the counter reload value dictates the sampling rate of PEBS. The counter can be configured for either a precise or non-precise event.

Architectural PEBS expands on the usability and applicability of legacy PEBS versions. The Output Buffer, a memory area where the processor dumps the CPU state, is physically addressed and configured via MSRs, making it virtualization-friendly for system-wide profiling (e.g., for hypervisors).

Architectural PEBS configuration is per counter in support of multi-tenant environments. The PEBS record is restructured, too, for a consistent yet expandable format for tools and enriched with additional state information (e.g., the XER group). Architectural PEBS ensures a state is not exposed across protection boundaries. It also improves the performance over previous PEBS versions using PMI-based sampling. Note that any cores using architectural PEBS will not support legacy PEBS, as it will have been deprecated.

22.10.1 Configuration

If CPUID.23H,0H:EAX[5] is set to 1, the processor supports Architectural PEBS. In this case, sub-leaves 04H and 05H of CPUID Leaf 23H indicate details of Architectural PEBS capabilities (see Chapter 21 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1). MSR enumeration of legacy PEBS is also modified in IA32_PERF_CAPABILITIES and IA32_MISC_ENABLE.

22.10.1.1 IA32_PERF_CAPABILITIES

Architectural PEBS requires changes to the IA32_PERF_CAPABILITIES MSR (address 345H):

- BASELINE[14] = 0.
- PEBS_TRAP[6], PEBS_ARCH_REG[7], and PEBS_TIMING_INFO[17] = 1.
- PEBS_RECORD_FORMAT[11:8] = 0xF (legacy PEBS is unavailable).
- PEBS_Output_PT_Available[16] = 0.

22.10.1.2 IA32_MISC_ENABLE

Architectural PEBS requires changes to the IA32_MISC_ENABLE MSR (address 1A0H). Specifically, the read-only field PEBS_UNAVAILABLE[12] is removed, but its value is kept as 1.

22.10.2 Behavior

The processor effectively stores the selected CPU state to memory when a PEBS event occurs, namely after a performance-monitoring counter overflows. The counter can be configured with a precise or non-precise performance-monitoring event. PEBS generates a record for precise events immediately after the instruction (as specified by the Instruction Pointer field of the Basic group) that experienced the event. For non-precise events, PEBS generates a record for some instruction when the counter overflows.

A PEBS record is always generated at an instruction boundary, reducing the need for expensive performance monitoring interrupt (PMI)-based sampling.

22.10.2.1 Counters

Section 22.2 describes general-purpose and fixed-function counters. Section 22.2.6 provides further details on the performance monitoring architecture. CPUID is used to enumerate the number and type of counters.

Reload values associated with individual counters are set in the lowest 32 bits of the extension registers, specifically IA32_PMC_GpN_CFG_C.RELOAD_VALUE[31:0] for general-purpose counters and IA32_PMC_Fxm_CFG_C.RELOAD_VALUE[31:0] for fixed-function counters, where n and m are the counter numbers.

Since the counters count up, the reload values are configured by storing the two's-complement negation first. For example, the value 1024 would be written as -1024 or 0xFFFFFC00 using 32 bits. On a reload, for a 48-bit Architectural performance-monitoring counter, the processor 1-extends this to 48 bits (0xFFFF_FFFFC00).

Furthermore, PEBS is enabled for a counter when the logical processor's CPL (Current Privilege Level) satisfies the counter configuration.

Architectural PEBS / Architectural PerfMon does not mandate a particular sequence for enabling counting. However, our clear recommendation is to enable IA32_PERF_GLOBAL_CTRL last / disable it first as it provides atomic starting/stopping of the counters.

To reiterate, the PEBS enable bits are distributed per counter, unlike the “global” register in model-specific PEBS.

22.10.2.2 Record Data

The CPU writes a PEBS Record for each counter that has overflowed at the instruction boundary. Each PEBS Record contains a header followed by a collection of record groups. The header contains information about the size of the PEBS record and a bit vector indicating the presence of selected record groups. Available record groups are the Basic group, the Memory Auxiliary group, the General-Purpose Registers group, the XSAVE-Enabled Registers group, the Last Branch Records group, and the Performance-Monitoring Counters group. The Basic group is present by default in PEBS records as it includes necessary data for processing a record. These groups are listed in Table 22-109 and described in detail in Section 22.10.4

Table 22-109. Architectural PEBS Configuration Locations

Register/Format	Bit Description								
	31	30	29	28:24	23:17	16:10	9:8	7:3	2:0
CPUID.23H.04H:EBX	0	AUX	GPR	0	XER	0	LBR	CNTR/A	0
	63	62	61	60:56	55:49	48:42	41:40	39:35	34:32
IA32_PMC_GpN_CFG_C IA32_PMC_Fxm_CFG_C	PEBS_EN	AUX	GPR	MBZ	XER	MBZ	LBR	CNTR/A	MBZ
	63	62	61	60:56	55:49	48:42	41:40	39:35	
PEBS Record Format	BASIC	AUX	GPR	RSVD	XER	RSVD	LBR	CNTR/ RSVD	RSVD

22.10.2.3 Programming PEBS

This section lists the programming guidelines¹ that system software should follow in the specified order. Then, it illustrates the behavior through an example.

22.10.2.3.1 Programming Guidelines at Initialization Phase

1. Disable counting globally.
2. Setup the PEBS output buffer via the IA32_PEBS_BASE and IA32_PEBS_INDEX registers.
3. Configure counter(s).²
 - a. Configure a performance-monitoring event into a counter.³
 - 1) Select a performance-monitoring event via the IA32_PMC_GpN_CFG_A MSR for general-purpose counters. This step does not apply for fixed-function counters.
 - 2) Set CPL filters via the IA32_PMC_GpN_CFG_A MSR for general-purpose counters or via the IA32_FIXED_CTR_CTRL MSR for fixed-function counters.

-
1. The first and last steps in both lists may rely on IA32_PERF_GLOBAL_CTRL. Architectural PEBS and Architectural Performance-Monitoring do not mandate a particular sequence for enabling counting. However, the recommendation is to enable IA32_PERF_GLOBAL_CTRL last or disable it first, as it provides atomic starting and stopping of the counters and ensures the least disruption to profiling.
 2. This step may be repeated for each counter that software wishes to generate a PEBS record, at initialization, or when another event is configured (e.g., event-counter multiplexing usage).
 3. The counter registers are accessed in the IA32_PMC_GpN_CTR MSR for general-purpose counter n, or in the IA32_PMC_Fxm_CTR MSR for fixed-function counter m.

- 3) Enable the counter locally.
- b. Configure PEBS for the same counter via the IA32_PMC_GPn/FXm_CFG_C MSR.¹
 - 1) Program a Reload value.
 - 2) Select the groups to be included in the PEBS records; see Table 22-109.
 - 3) Set the PEBS_EN bit.
- c. Initialize the counter register with its Reload value.
4. Re-enable counting globally.

22.10.2.3.2 Programming Guidelines at Runtime (e.g., Inside the PMI Interrupt Service Routine)

1. Disable counting globally.
2. Handle PEBS.
 - a. Drain data from the PEBS buffer; for example, copy-over data to another buffer. This is the most-common case for which PEBS-triggered PMI is invoked.
 - b. Update IA32_PEBS_INDEX[WR_OFFSET] to the start of the PEBS buffer, if needed.
 - c. Clear IA32_PERF_GLOBAL_STATUS[ARCH_PEBS] via the IA32_PERF_GLOBAL_STATUS_RESET MSR if the PEBS buffer Threshold is enabled and/or Clear IA32_PEBS_INDEX[Full].
3. Handle counter n overflow.
 - a. Set the counter register to its Reload value. Retrieve n, the counter index, via one of the following methods:
 - 1) The Applicable Counter field of the Basic group of most recent record(s) in step 2a above.
 - 2) Traversing the counter registers' values.
 - b. In some usages, software may need to save the current value of counter n, prior to reloading the counter value.
4. Re-enable counting globally.

For general-purpose counters, PEBS is enabled when IA32_PMC_GPn_CFG_C.PEBS_EN && IA32_PERF_GLOBAL_CTRL.EN[n] && IA32_PMC_GPn_CFG_A.CNTR_EN is true, where n is the counter index.

For fixed-function counters, PEBS is enabled when IA32_PMC_FXm_CFG_C.PEBS_EN && IA32_PERF_GLOBAL_CTRL.EN[32+m] && IA32_FIXED_CTR_CTRL.[USR|OS]m is true, where m is the counter index.

Furthermore, PEBS is enabled for a counter when the logical processor's Current Privilege Level (CPL) satisfies the counter configuration. The PEBS enable bits are distributed per counter, unlike the "global" register in legacy PEBS.

Example 22-3. Programming PEBS

To get the records, as shown in Table 22-74, software should use the following steps:

1. Disable counting globally by writing zero to the IA32_PERF_GLOBAL_CTRL MSR (address 38FH).
2. Set up IA32_PEBS_BASE and IA32_PEBS_INDEX registers.
3. For the general-purpose counter 1:
 - a. Configure the event BR_INST_RETIRED.ALL_BRANCHES, which counts retired branches.
 - b. Set the PEBS_EN[63] bit in the IA32_PMC_GP1_CFG_C MSR (address 1907H). This enables PEBS.
 - c. Put the 32-bit reload value into IA32_PMC_GP1_CFG_C.RELOAD_VALUE.
 - d. There is no need to set other bits for the optional groups since the Basic group is included by default in PEBS records.
4. For the fixed-function counter 1:

1. Details on the referenced IA32_* registers can be found in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

- a. Locally enable it by setting the CPL bits IA32_FIXED_CTR_CTRL.[USR1|OS1]. This counter counts CPU_CLK_UNHALTED.THREAD – Unhalted Core Cycles.
 - b. Set the PEBS_EN[63] bit in the IA32_PMC_FX1_CFG_C MSR (address 1987H). This enables PEBS.
 - c. Put the 32-bit reload value into IA32_PMC_FX1_CFG_C.RELOAD_VALUE.
 - d. Set the AUX[62] bit and GPR[61] bit in the IA32_PMC_FX1_CFG_C MSR. This includes the AUX and GPR groups in the record.
5. Re-enable counting globally by setting bits 1 and 33 in the IA32_PERF_GLOBAL_CTRL MSR to start counting on both counters. See Chapter 2 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for details of the MSR.

When the general-purpose counter and the fixed-function counter overflow, the two records are written to memory.

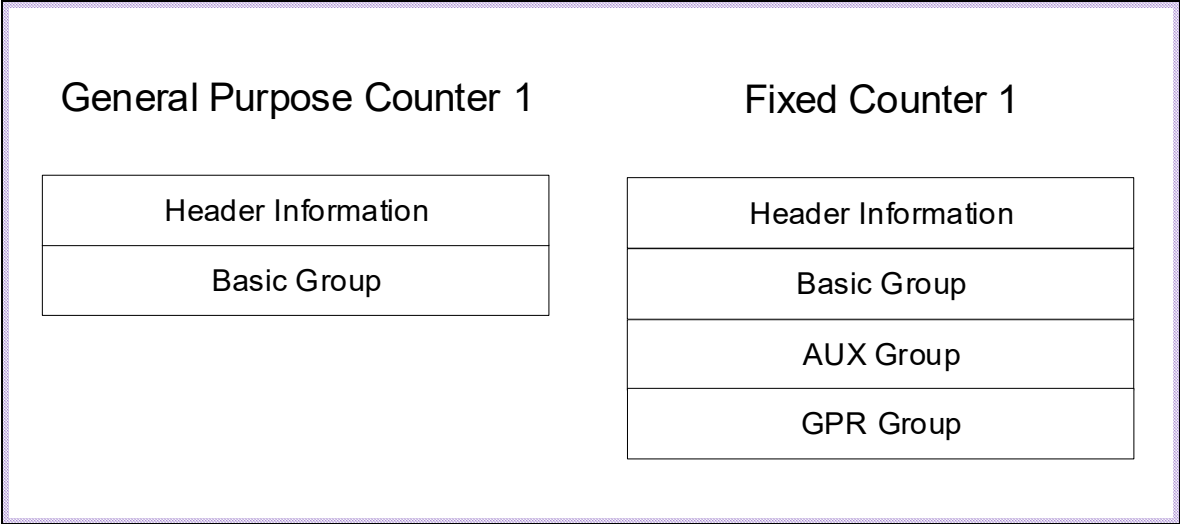


Figure 22-74. Example PEBS Record

22.10.3 Model-Specific Registers (MSRs)

The Architectural PEBS MSRs are IA32_PEBS_BASE, IA32_PEBS_INDEX, IA32_PMC_GPn_CFG_C, and IA32_PMC_FXm_CFG_C.

22.10.3.1 IA32_PEBS_BASE MSR

Figure 22-75 shows the layout of the IA32_PEBS_BASE MSR, the Output Buffer Base Register MSR.

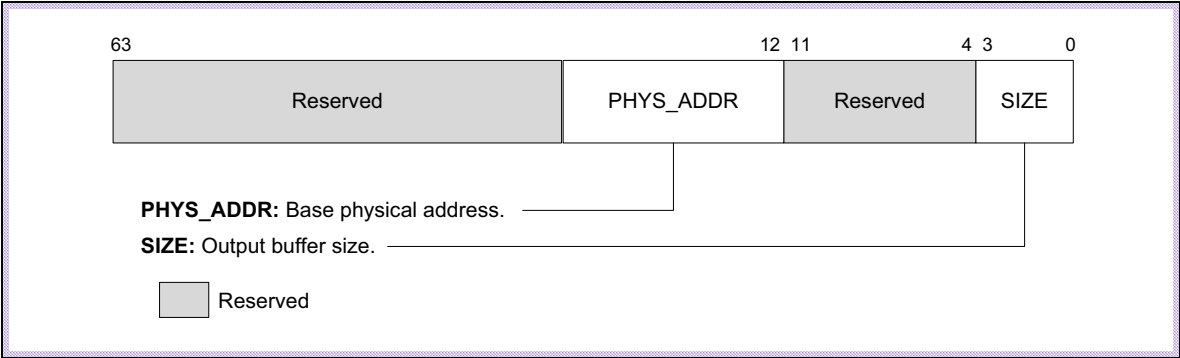


Figure 22-75. IA32_PEBS_BASE Register

The functions of the flags and fields are:

- **SIZE**, bits 3:0 – Specifies the output buffer size in powers of 2, where buffer size = 4 KB * 2^{SIZE}, e.g., SIZE = 7 results in a 512 KB output buffer and SIZE = 9 results in a 2 MB output buffer. The maximum SIZE value is 15, or 128 MB output buffer.
- **PHYS_ADDR**, bits (MAXPHYADDR – 1):12 – Specifies the value of the base physical address of a single, contiguous physical output region.

The MSR is initialized to all zeros on reset.

22.10.3.2 IA32_PEBS_INDEX MSR

Figure 22-76 shows the layout of the IA32_PEBS_INDEX MSR, the Output Buffer Index Register MSR.

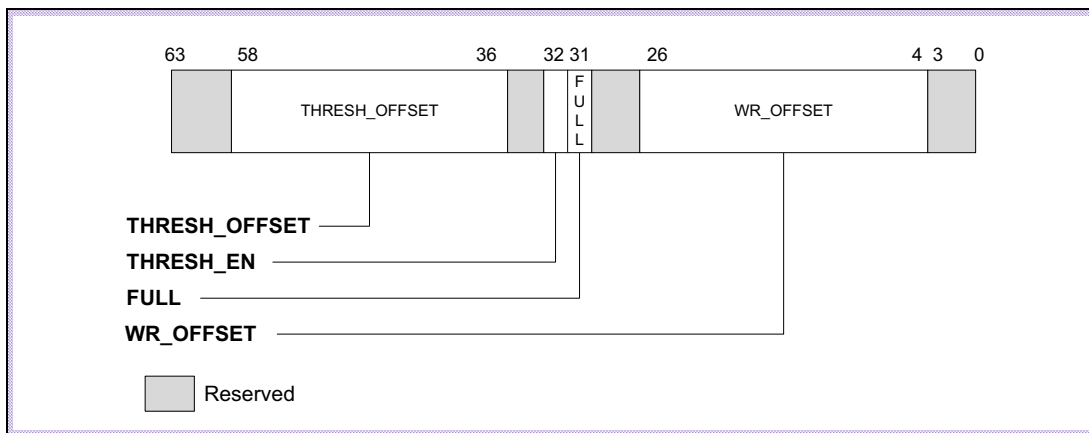


Figure 22-76. IA32_PEBS_INDEX Register

The functions of the flags and fields are:

- **WR_OFFSET**, bits 26:4 – Specifies the value of the next-byte-to-write offset from IA32_PEBS_BASE.PHYS_ADDR.
- **FULL**, bit 31 – When set, the output buffer is full, and no further records will be generated.
- **THRESH_EN**, bit 32 – When set, the PMI threshold is enabled.
- **THRESH_OFFSET**, bits 58:36 – Specifies the value of bits [26:4] of the PMI threshold offset from the output buffer base.

The MSR is initialized to all zeros on reset.

22.10.3.3 IA32_PMC_GPn_CFG_C and IA32_PMC_FXm_CFG_C MSRs

Table 22-110 shows the layout of the Performance Event Select Extended Register MSRs. The IA32_PMC_GPn_CFG_C MSR is the Event Select Extended register for a general-purpose counter, while the IA32_PMC_FXm_CFG_C MSR is the Event Select Extended register for a fixed-function counter, where n and m are the counter numbers.

Table 22-110. IA32_PMC_GPn_CFG_C and IA32_PMC_FXm_CFG_C MSRs

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1903H, 1907H, ..., 1903H+(4*n), 6403, 6407, ..., 6403 +(4*n)	IA32_PMC_GPn_CFG_C	
Extended Performance Event Selector for GP Counter x (R/W) For general-purpose counters, the IA32_PMC_GPn_CFG_C MSR is available if either CPUID.23H.05H:EAX[n] or CPUID.03H.02H:EAX[n] is set. Reserved bits must be zero.		
31:0	RELOAD_VALUE (R/W) Contains the reload value to be loaded into the associated counter by Auto Counter Reload. It will be 1-extended to 48 bits.	
34:32	Reserved.	
35	ALLOW_IN_RECORD (R/W) If 1, indicates that this counter's value will be included in the PEBS record if the associated CNTR subgroup is enabled.	
38:36	CNTR (R/W) When any bit is set, the counter group will be included in the PEBS record. Bit 36: If 1, the general-purpose counters subgroup will be included. Bit 37: If 1, the fixed-function counters subgroup will be included. Bit 38: If 1, the performance metrics subgroup will be included and will be reset	
39	Reserved.	
41:40	LBR (R/W) Identifies the number of Last Branch Records (LBRs) to record. The encodings of this field are as follows: 00 – LBR group will not be recorded. The group is disabled. 01 – 8 LBRs will be recorded. 10 – 16 LBRs will be recorded. 11 – The number of LBRs to be recorded is IA32_LBR_DEPTH.DEPTH.	
48:42	Reserved.	
55:49	XER (R/W) When set, enables particular XSAVE-Enabled Registers to be included in the PEBS record. The encodings for the register subgroup are: - Bit 49 – SSER: Record XMM0-XMM15. - Bit 50 – YMMHIR: Record the upper 128 bits of YMM0-YMM15. - Bit 51 – EGPR: Record R16-R31. - Bit 52 – Reserved and must be zero. - Bit 53 – OPMASKR: Record KO-K7. - Bit 54 – ZMMHIR: Record the upper 256 bits of ZMM0-ZMM15. - Bit 55 – Hi16ZMMR: Record ZMM16-ZMM31.	
60:56	Reserved.	
61	GPR (R/W) If 1, enables the General-Purpose Registers Group to be included in the PEBS record.	
62	AUX (R/W) If 1, enables the Auxiliary Group to be included in the PEBS record.	
63	PEBS (R/W) If 1, PEBS is enabled for this counter.	

Table 22-110. IA32_PMC_GPn_CFG_C and IA32_PMC_FXm_CFG_C MSRs (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1983H, 1987H, ..., 1983H+(4*m), 6531, 6535, ..., 6531 +(4*m)	IA32_PMC_FXm_CFG_C	
Extended Performance Event Selector for Fixed-Function Counter x (R/W) For fixed-function counters, the IA32_PMC_FXm_CFG_C MSR is available if either CPUID.23H.05H:ECX[m] or CPUID.23H.02H:EBX[m] is set. Reserved bits must be zero.		
63:0	This MSR has identical bit definitions to the IA32_PMC_GPn_CFG_C MSR.	

The functions of the flags and fields are:

- **RELOAD_VALUE**, bits 31:0 – Selects a 32-bit value loaded into IA32_PMC_GPn_CTR when a PEBS record is generated due to counter n. For a 48-bit counter, the value will be 1-extended into a 48-bit value.
- **ALLOW_IN_RECORD**, bit 35 – When set, indicates that this counter's value will be included in the PEBS record if the associated CNTR subgroup is enabled.
- **CNTR**, bits 38:36 – When any bit is set, the counter group will be included in the PEBS record. If bit 36 is set, the general-purpose counters subgroup will be included. If bit 37 is set, the fixed-function counters subgroup will be included. If bit 38 is set, the performance metrics subgroup will be included and reset.
- **LBR**, bits 41:40 – Identifies the number of Last Branch Records (LBRs) to record. The encodings of this field are as follows:
 - 00 – LBR group will not be recorded. The group is disabled.
 - 01 – 8 LBRs will be recorded.
 - 10 – 16 LBRs will be recorded.
 - 11 – The number of LBRs to be recorded is IA32_LBR_DEPTH.DEPTH.
- **XER**, bits 55:49 – When set, enables particular XSAVE-Enabled Registers to be included in the PEBS record. The encodings for the register subgroup are:
 - Bit 49 – SSER: Record XMM0-XMM15.
 - Bit 50 – YMMHIR: Record the upper 128 bits of YMM0-YMM15.
 - Bit 51 – EGPR: Record R16-R31.
 - Bit 53 – OPMASKR: Record K0-K7.
 - Bit 54 – ZMMHIR: Record the upper 256 bits of ZMM0-ZMM15.
 - Bit 55 – Hi16ZMMR: Record ZMM16-ZMM31.
- **GPR**, bit 61 – When set, this field enables the General-Purpose Registers group to be included in the PEBS record.
- **AUX**, bit 62 – When set, this field enables the Auxiliary group to be included in the PEBS record.
- **PEBS_EN**, bit 63 – When set, PEBS is enabled for this counter.

On reset, the MSR is initialized to all zeros.

22.10.3.4 Changes from Legacy PEBS

The following MSRs are removed and generate a #GP exception on any attempt to read or write them:

- IA32_PEBS_ENABLE
- MSR_PEBS_DATA_CFG

Architectural PEBS does not use the IA32_DS_AREA MSR, but it is maintained for BTS use.

Bits 32, 36, 40, and 44 of IA32_PMC_FXm_CFG.C.ADAPTIVE_RECORD_CTRm will report 0 on a RDMSR and will generate a #GP exception on an attempt to set them. The same results occur with IA32_PMC_GPn_CFG.Adaptive PEBS for all general-purpose counters. See Section 22.9.2.1 for more information.

The IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_STATUS_RESET, and IA32_PERF_GLOBAL_STATUS_SET MSRs are modified by adding a bit, ARCH_PEBS[54]:

- The processor clears IA32_PERF_GLOBAL_STATUS[54]=0 on a software write of 1 to IA32_PERF_GLOBAL_STATUS_RESET[54].
- The processor sets IA32_PERF_GLOBAL_STATUS[54]=1 on a software write of 1 to IA32_PERF_GLOBAL_STATUS_SET[54].

See Section 22.2.4.2 for more information.

22.10.4 Records and Groups

All records begin with a 16-byte header, shown in Figure 22-77, which includes the 8-byte PEBS record format. The record format contains the following:

- The bit vector indicating the groups in the record
- The record size, which may include end padding (see Section 22.10.4.7)
- Other indicator bits

The remaining 8 bytes in the header are reserved.

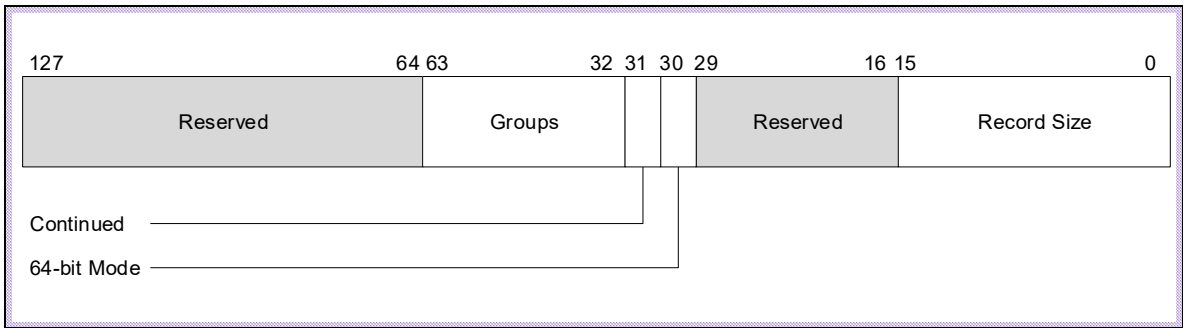


Figure 22-77. PEBS Record Format

The record size is a 16-bit field containing the size of the generated record (the header plus the groups) in bytes, with a maximum size of 64 kilobytes. A record always starts at a 16-byte aligned address. The size may include padding at the end for alignment reasons. The padding bytes would be stale and not written by the processor.

The Continued bit indicates that the record has additional groups in subsequent record fragments. See Section 22.10.4.7 for an example.

A record continues in the architecturally defined group order: Basic group, Memory Auxiliary group, General-Purpose Registers group, XSAVE-Enabled Registers group, Last Branch Records group, and finally, Performance-Monitoring Counters group. The record is also compacted to place only those groups indicated in the bit vector contiguously.

All groups are aligned along 16-byte boundaries and multiples of 16 bytes. Some groups have reserved fields and subfields to preserve alignment requirements. Most groups have a predetermined size, though some are variable.

A processor may dump groups/fields in any temporal order, not necessarily per their offset.

22.10.4.1 Basic Group

The Basic group is present in all complete PEBS records (see Section 22.10.4.7). The group size is 48 bytes and consists of the Instruction Pointer, Applicable Counter, Time-Stamp Counter, and Retire Latency fields. The final two fields in the group, each consisting of 8 bytes, are reserved for future growth.

Table 22-111 lists the Basic group's fields and the offset from the start of the Basic group. Each field is 64 bits wide.

Table 22-111. Basic Group Fields

Field Name	Byte Offset
Instruction Pointer	00H
Applicable Counter	08H
Time-Stamp Counter	10H
Retire Latency	18H
Reserved	20H
Reserved	28H

22.10.4.1.1 Instruction Pointer

The Instruction Pointer field holds the RIP register value¹ of the instruction that retired immediately before emitting the PEBS record. This field is sometimes historically referred to as “EventingIP.”

For precise events, this is the instruction that triggered PEBS record generation. For non-precise events, this is some instruction when the counter overflows without necessarily stopping at the causing instruction.

If a record was generated immediately after execution just became “in context” for a given counter, then the Instruction Pointer of that record is considered “out of context.” A -1 value denotes an out-of-context Instruction Pointer. This behavior avoids leaking information across contexts.

Note that the Instruction Pointer field differs from the RIP field in the General-Purpose Registers group, which records the instruction pointer of the next instruction to be executed after record generation. Also, when not in 64-bit mode, the upper 32 bits will be zero.

22.10.4.1.2 Applicable Counter

This field indicates which counter(s) triggered the generation of the PEBS record.

Figure 22-78 shows the mapping of the various one-hot counter bits in the PEBS record. IA32_PMC[i-1:0] is the bit vector of general purpose counters, where IA32_PMC0 starts at bit 0, and i is the maximum general purpose counter ID supported. IA32_FIXED_CTR[32+j-1:32] is the bit vector of fixed counters, where IA32_FIXED_CTR0 starts at bit 32, and j is the maximum fixed counter ID supported. Bits that do not correspond to counters identified in Section 22.2.9 are reserved.

1. RIP register without the CS register base.

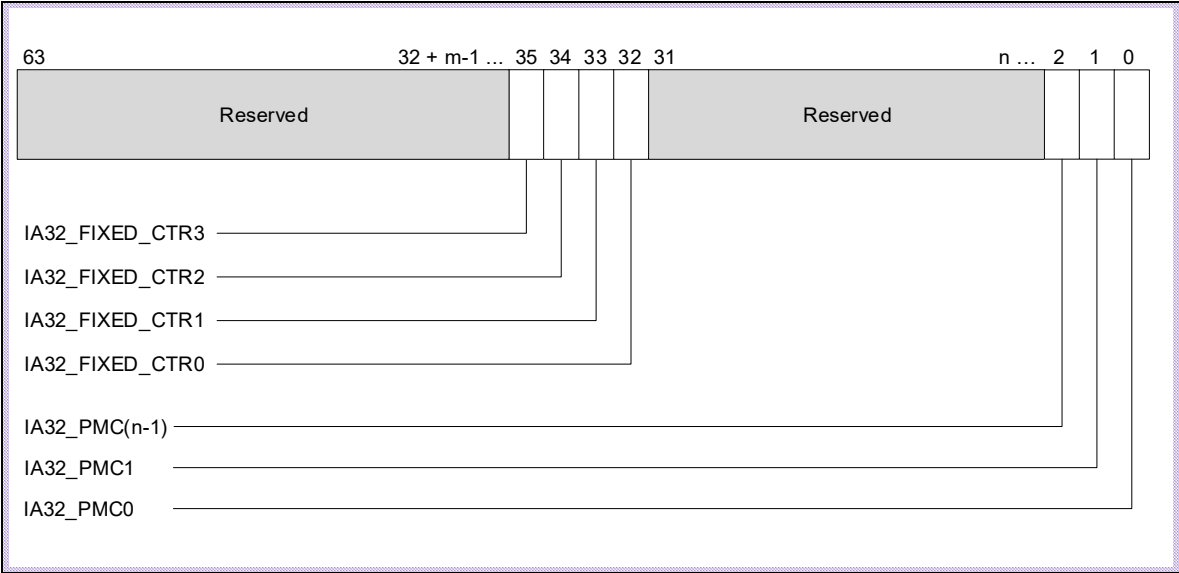


Figure 22-78. Bit Mapping of Applicable Counters in the PEBS Record

22.10.4.1.3 Time-Stamp Counter

The time-stamp counter is a 64-bit counter set to 0 following a processor reset. After a reset, the counter increments even when the HLT instruction or the external STPCLK# pin halts the processor.

The time-stamp counter will reflect the value of the counter when the record is generated, and it will include the offset and scale adjustments even if RDTSC Exiting is enabled (i.e., no exit taken). It is the same as Intel Processor Trace.

For more details, see Section 20.17, “Time-Stamp Counter,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B.

22.10.4.1.4 Retire Latency

The Retire Latency field in the record will consist of:

- **Retire Latency field**, bits 15:0 – Indicates the elapsed cycles between the retirement of the architecturally-visible instruction that caused PEBS and the prior instruction retirement. The measurement reflects core unhalted cycles (at the pace of Fixed-Function Counter 1) and is reported for any PEBS, regardless of whether precise or non-precise events are programmed. The count saturates at 216-1.
- **Retire Latency Valid flag**, bit 16 – A valid bit for the Retire Latency field. When set, the Retire Latency field holds a valid value.
- Reserved field, bits 63:17 – Reserved for future use.

22.10.4.2 Auxiliary Group

The Auxiliary (AUX) group is home to custom fields, often without ties to the state of a particular processor architectural feature and without architectural registers. The AUX group size is 64 bytes, consisting of 8 fields of 8 bytes each. The first 8-byte field is the Memory Access Address, followed by seven 8-byte reserved fields, as shown in Table 22-112. A named reserved field may have processor-dependent implementation.

Table 22-112. Auxiliary Group Fields

Field Name	Byte Offset
Memory Access Address	00H
Reserved	08H
Reserved	10H
Reserved	18H
Reserved	20H
Reserved (Memory Auxiliary Information)	28H
Reserved (Memory Access Latency)	30H
Reserved (TSX Auxiliary Information)	38H

If CPUID.23H.04H:EBX.AUX[30] = 1, then the Auxiliary group is available. If AUX[62] in the counter's Event Select Extended register is set, then the Auxiliary group will be included when that particular counter generates a PEBS record.

If PEBS Record Format.AUX[62] = 1 in the record itself, then the Auxiliary group is included.

All fields are populated in a model- and event-specific manner. None of the architectural events are architecturally specified to populate these fields, including the Memory Access Address field.

If the Memory Access Address population is not indicated in the model-specific event list, the field contents are reserved.

22.10.4.2.1 Memory Access Address

The Memory Access Address (MAA) is the 64-bit canonical linear address¹ of memory accessed by a load or store access of an instruction. The location of the MAA field is architectural, but which events update it and the address saved is model-specific. Unlike legacy PEBS, Architectural PEBS enables one instruction to generate multiple records with different MAA fields. For example, MOVDIR* instructions have distinct load and store addresses.

The MAA will be the address of a linear access performed by the instruction immediately preceding the PEBS event (at EventingIP, or macrofused² to it), or zero. The MAA will never expose access from any of these cases:

- A filtered out CPL, even for ring transitions
- Before any counters are enabled, e.g., a VMM address after a VM entry enables counters
- Inside a production/opt-out enclave (no PEBS records are taken during a production enclave, and PEBS records taken after EEXIT/AEX will have MAA=0)

For instructions with multiple linear load accesses, or multiple linear store accesses, which one (if any) is recorded is model-specific. Consult the online event list for more details on the MAA behavior for a particular processor: <https://perfmon-events.intel.com/>.

NOTE

For MSR flows (VMX MSR list, WRMSRLIST) that can disable and re-enable counters, it is possible for an MAA recorded before the disable to persist. Intel advises against such uses.

22.10.4.3 General-Purpose Register Group

The General-Purpose Register group contains 20 fields, each being 64 bits wide, for a total of 160 bytes. The group contains RFLAGS, RIP, RAX, RCX, etc., as well as the Shadow Stack Pointer (SSP). The SSP will be zero on implementations that do not support CET or when CET is not in use.

1. No Linear Address Masking (LAM) bits are included.
2. Macrofusion merges two instructions to a single micro-op. See Section 3.4.2.2, "Optimizing for Macrofusion," in the Intel® 64 and IA-32 Architectures Optimization Reference Manual: Volume 1.

If CPUID.23H.04H:EBX.GPR[29] = 1, then the General-Purpose Register group is available. If IA32_PMC_GPn_CFG_C.GPR[61]=1 or IA32_PMC_FXm_CFG_C.GPR[61] = 1, then the record will include the General-Purpose Register group when the counter generates a PEBS record. In the PEBS record, if PEBS Record Format.GPR[61] = 1, the General-Purpose Register group is included.

Table 22-113. General-Purpose Register Group Fields

Field Name	Byte Offset
RFLAGS	00H
RIP	08H
RAX	10H
RCX	18H
RDX	20H
RBX	28H
RSP	30H
RBP	38H
RSI	40H
RDI	48H
R8	50H
...	...
R15	88H
SSP	90H
Reserved	98H

The fields are sized to support 64-bit mode. Modes are checked at the time of PEBS record generation.

When not in 64-bit mode, the upper 32 bits of the RAX through RDI fields will be unmodified. Furthermore, fields for registers that “do not exist” are left unmodified, similar to how XSAVE handles vector registers. Space is not compacted away. Registers R8-R15 will be unmodified. This is consistent with model-specific PEBS behavior.

When not in 64-bit mode, fields for registers that do not exist are left unmodified. Core legacy model-specific PEBS wrote zero.

PEBS enable is not impacted by transitions into/out of 64-bit mode. Some legacy PEBS versions used to disable it. The indicator bit in PEBS Record Format.64BIT_MODE[30] will be set to 1 for 64-bit mode and 0 for other modes.

22.10.4.4 XSAVE-Enabled Registers Group

The XSAVE-Enabled Registers (XER) group is a super-group for user-level state with XSAVE architectural support, i.e., those supported by the XSAVEC instruction. Five groups (or sub-groups) are available initially (see Table 22-114). Additional groups are likely to be supported in future generations. The supported groups depend on the ISA support of the processor for SSER, YMMHIR, EGPR, OPMASKR, ZMMHIR, and Hi16ZMMR.

CPUID.23H.04H:EBX.XER[23:17] indicate which components of the XER super-group are available. The bits IA32_PMC_GPn_CFG_C.XER[55:49] and IA32_PMC_FXm_CFG_C.XER[55:49] select which groups of XER to include when the counter generates a PEBS record. The select bit in the *CFG_C MSRs is writeable if that group is available in CPUID.

The bits PEBS Record Format.XER[55:49] indicate which components of the XER group are included in the PEBS record. This field is set to 0 if CR0.TS = 1 or CR4.OSXSAVE = 0. Otherwise, a bit is set only if a group is selected in the CFG MSR and the XCR0 bit corresponding to the group is set. Group sizes are listed in Table 22-114.

Table 22-114. XSAVE-Enabled Registers (XER) Group Fields and Sizes

Field Name	Registers Used	Size
XSTATE_BV	XINUSE for groups included in PEBS Record Format.XER[55:49]	8 B
Reserved	Reserved	8 B
SSER	XMM0-XMM15	16 regs * 16 B = 256 B
YMMHIR	Upper 128 bits of YMM0-YMM15	16 regs * 16 B = 256 B
EGPR	R16-R31	16 regs * 8 B = 128 B
OPMASKR	K0-K7	8 regs * 8 B = 64 B
ZMMHIR	Upper 256 bits of ZMM0-ZMM15	16 regs * 32 B = 512 B
Hi16ZMMR	ZMM16-ZMM31	16 regs * 64 B = 1024 B

The first eight bytes include the XSAVES instruction's XSTATE_BV bit vector (reflecting INIT optimization). This field is in XCR0 format.

Memory space in the output buffer is allocated for the groups that are included in PEBS Record Format.XER[55:49]. The memory space is allocated but not written if that group is either not in use (that is, in the INIT state) or is disabled by XFD (eXtended Feature Disable). (In this case the corresponding bit in XSTATE_BV bitmap is set to 0.) The memory space is allocated and written when all conditions are met (that is, the group is selected, enabled by XSAVE, and is in use).

Outside 64-bit mode, vector registers beyond XMM7/YMM7/ZMM7 are not accessible. Thus, the memory space is allocated but not written for those registers.

Processors that do not support a particular group, e.g., ZMMHIR, will have the corresponding bit clear in CPUID, and attempting to set the select bit will cause a #GP fault.

22.10.4.5 Last Branch Record Group

Last Branch Records (LBRs) enable the recording of the software path history by logging taken branches and other control flow transfers within processor registers. See Section 20.5, "Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors)," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, for information about last branch records.

Table 22-115 lists the LBR fields.

Architectural PEBS can include a subset of the architectural LBRs available. Software may, independent to PEBS, reduce the number of active LBR records from the maximum using the IA32_LBR_DEPTH MSR.

The bits IA32_PMC_GpN_CFG_C.LBR[41:40] and IA32_PMC_Fxm_CFG_C.LBR[41:40] indicate the number of LBR records to include in the Last Branch Record group. The encodings of this field are as follows:

- 00 – LBR group will not be recorded. The group is disabled.
- 01 – 8 LBRs will be recorded.
- 10 – 16 LBRs will be recorded.
- 11 – Variable size. The number of LBRs to be recorded is defined by IA32_LBR_DEPTH.DEPTH.

The bits PEBS Record Format.LBR[41:40] indicate the number of LBR records in the group.

Table 22-115. Last Branch Record Group Fields

Field Name	Byte Offset
Reserved	00H
IA32_LBR_CTL	08H
IA32_LBR_DEPTH	10H
IA32_LER_FROM_IP	18H

Table 22-115. Last Branch Record Group Fields

Field Name	Byte Offset
IA32_LER_TO_IP	20H
IA32_LER_INFO	28H
IA32_LBR_0_FROM_IP	30H
IA32_LBR_0_TO_IP	38H
IA32_LBR_0_INFO	40H
...	...
IA32_LBR_[Num_LBR-1]_FROM_IP	30H+[Num_LBR-1]*24
IA32_LBR_[Num_LBR-1]_TO_IP	38H+[Num_LBR-1]*24
IA32_LBR_[Num_LBR-1]_INFO	40H+[Num_LBR-1]*24

The group size is 6*8 bytes + Num_LBR*24 bytes. Software must look at IA32_LBR_DEPTH recorded in the group. If a fixed size was requested in the IA32_PMC_GPn_CFG_C or IA32_PMC_FXm_CFG_C register, IA32_LBR_DEPTH recorded in the group will indicate how many LBR entries in the group are valid (meaningful). If a variable size was requested in the IA32_PMC_GPn_CFG_C or IA32_PMC_FXm_CFG_C register, IA32_LBR_DEPTH recorded in the group will indicate the size of the variable portion of the LBR group.

The LBR array is preceded in the record by five additional LBR- and LER-related fields. Should a fixed number of Num_LBRs be requested, but LBR_DEPTH is set lower, entries beyond LBR_DEPTH will be unmodified. IA32_LBR_DEPTH is included in the LBR group in a specific location.

It is possible for LBRs to be in system wide mode (IA32_LBR_CTL[OS=1, USR=1]) while PEBS is in user mode. PEBS will not attempt to filter out any non-user data from the LBRs, including LERs. It is up to the software to ensure that if it allows LBRs in PEBS, that the filtering conditions are appropriate.

The size of a single LBR entry is 3 fields * 64 bits, or 24 bytes. The contents of each field is determined by the architectural LBR feature.

22.10.4.6 Performance Counters Group

Performance Counters enable performance monitoring events to be counted during processor execution. The Counters group (CNTR) enables users to snapshot the current values of select performance monitoring counters into the PEBS record, including counters other than the one that triggered the generation of the PEBS record. General purpose (GP), fixed function (FIXED), and performance metrics (METRICS) are covered by the Counters group. Table 22-116 provides the Counter group details.

The bits CPUID.23H.04H:EBX.CNTR[7:4] indicate that the Counter group component subgroups are available.

IA32_PMC_GPn_CFG_C.CNTR[39:36] and IA32_PMC_FXm_CFG_C.CNTR[39:36] indicate whether to include the Counter group subgroups when this counter generates a PEBS record. Any set bit will cause the Counter group header to be included in the record.

The bits PEBS Record Format.CNTR[39:36] indicate which counter group subgroups were included in the record. If any of these bits are set, the PEBS record will include the 16-byte group header. The group header contains four 32-bit vectors, one vector per subgroup, indicating which counters are included in each subgroup. Individual counters must opt in.

CPUID.23H.04H:EBX.ALLOW_IN_RECORD[3] indicates the IA32_PMC_GPn_CFG_C.ALLOW_IN_RECORD and IA32_PMC_FXm_CFG_C.ALLOW_IN_RECORD bits are available. Software should set IA32_PMC_GPn_CFG_C.ALLOW_IN_RECORD[35] and IA32_PMC_FXm_CFG_C.ALLOW_IN_RECORD[35] to include the counter's value in the PEBS record when a record is generated that includes the associated subgroup.

The ALLOW_IN_RECORD, METRICS, FIXED, and GP bits within the extension registers allow software to control which counters actually appear in the PEBS record.

Note that the bit PEBS Record Format.ALLOW_IN_RECORD[35] is reserved.

Table 22-116. Counter Group Details

Bit	Reserved	METRICS Subgroup	FIXED Subgroup	GP Subgroup	ALLOW_IN_RECORD
CPUID.23H.04H:EBX	7	6	5	4	3
IA32_PMC_GPn_CFG_C	39	38	37	36	35
IA32_PMC_FXm_CFG_C					
PEBS Record Format	39	38	37	36	35
Group Header Bits	127:96	95:94	63:32	31:0	N/A

Consider the following example using general-purpose counters, where:

- IA32_PMC_GP0_CFG_C.{ALLOW=1, GP=1}
- IA32_PMC_GP1_CFG_C.ALLOW=1
- IA32_PMC_GP2_CFG_C.ALLOW=0

If general-purpose counter 0 generates a PEBS record, it includes general-purpose counter 0 and general-purpose counter 1 (but not general-purpose counter 2 because its ALLOW bit was clear).

If general-purpose counter 1 or general-purpose counter 2 generates a PEBS record, it does not include any counters because all METRICS, FIXED, and GP bits were clear for general-purpose counter 1 and general-purpose counter 2.

22.10.4.7 Fragmented Records

A PEBS record can be split into *n* fragments that comprise the complete record. Fragmented records can help mitigate the contiguous buffer when PEBS2GPA is used. See Section 22.10.5.1 for information about PEBS2GPA and Chapter 31, “VMX Support for Address Translation” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 3C, for EPT details.

Each fragment of the complete record has a header and a subset of the groups. The first *n*-1 fragments will have their Continued[31] bit set to 1 to indicate that the record has additional groups in subsequent record fragments. The last fragment will have its Continued bit clear.

If there is empty space between record fragments, then either the Record Size field may indicate padding (i.e., having a value larger than the required size as described in Section 22.10.4) or a NULL record will be inserted. A NULL record is an empty record where only the Record Size field is non-zero (group indicators, the Continued bit, and other fields are clear).

For example, suppose that a complete record consists of two fragments, record 4 and record 5, which cross an EPT page boundary as shown in Figure 22-79. In the top portion of Figure 22-79, record 4 includes some of the requested groups and padding to abut the end of the page, and it has its Continued bit set to 1. In the lower part of Figure 22-79, record 4 includes some of the requested groups with no padding (it has its Continued bit set to 1) and the Null Record is at the end of the page. Record 5 contains the remaining groups in both instances, with the Continued bit clear.

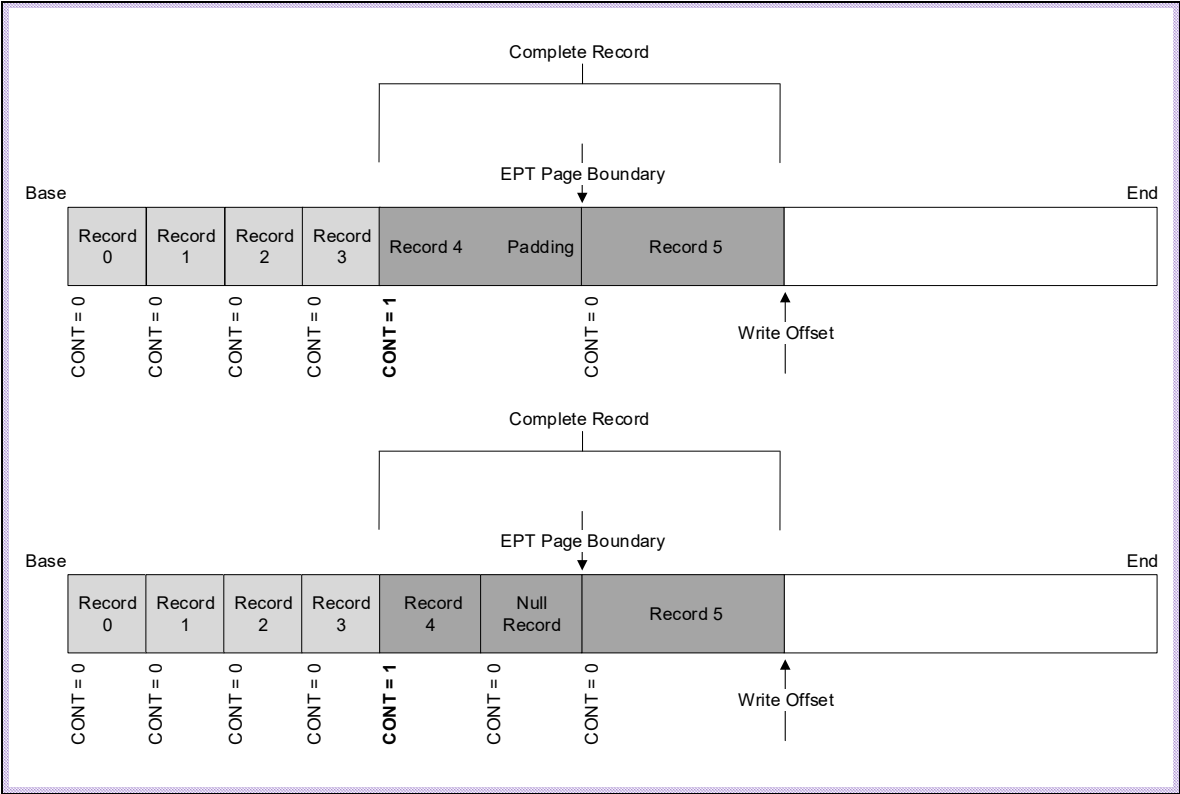


Figure 22-79. Example of a Fragmented Record

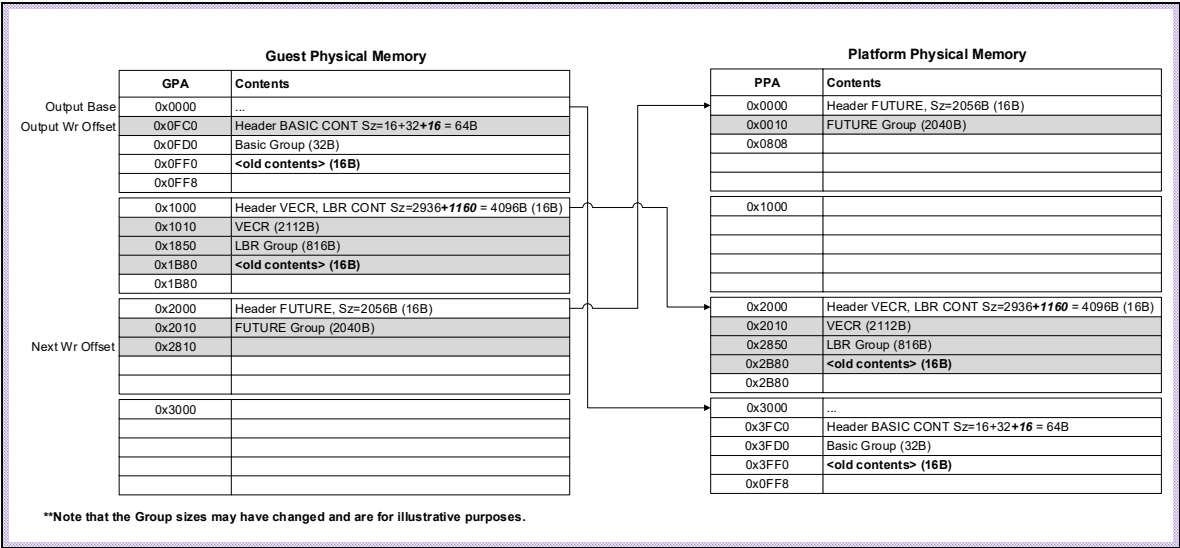


Figure 22-80. EPT Page Boundary Example with Included Padding

22.10.5 Interactions with Other Processor Features

22.10.5.1 Virtual Machine Extension (VMX)

By default, PEBS operation persists across VMX transitions. However, VMCS fields have been added to enable constraining PEBS usage to within VMX non-root operation only. See details in Table 22-117.

Table 22-117. PEBS VMCS Fields

Name	Type	Bit	Behavior
PEBS Uses Guest Physical Addresses	Tertiary Processor-Based VM-Execution Control	12 ¹	This bit determines whether the PEBS buffer uses guest physical addresses.

NOTES:

1. Definitions of Tertiary Processor-Based VM-Execution Controls. Software should consult the VMX capability MSR, IA32_VMX_PROCBASED_CTL3, to determine if this bit may be set. See the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D, Appendix A.3.4.

To enable the guest-only PEBS use case, a VMM should set the "PEBS Uses Guest Physical Addresses" VMCS execution control, and allocate the PEBS buffer using guest physical addresses. Additionally, counters should be enabled only in VMX non-root operation for that guest. In other words, on VM exit from the guest, the counters should be stopped, and on VM entry back to the guest, the counters should be resumed. This ensures that the counters and the PEBS record information belong to that guest.

In this use case, the guest owns counter/PEBS configuration, manages the PEBS output buffer, and switches the counter/PEBS state across applications inside that guest. The host can simply pass through the PEBS and counter management handling to that guest as illustrated by Figure 22-81.

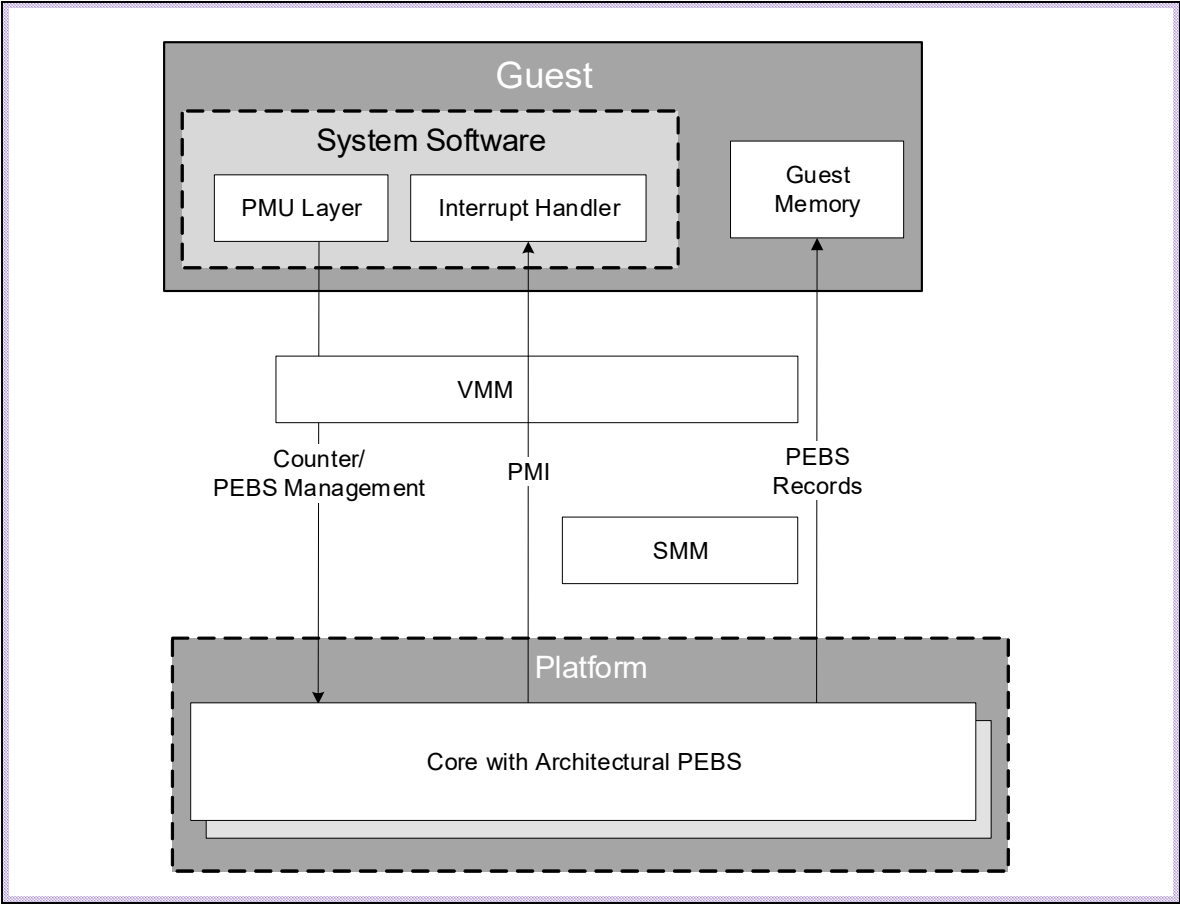


Figure 22-81. Guest-Only PEBS: Diagram of Interactions Across Entities

To enable the system-wide PEBS use case, a VMM should keep the PEBS Uses Guest Physical Addresses bit clear and allocate the PEBS buffer using physical addresses. Additionally, counters configured for PEBS should always be enabled, possibly hidden from guests. In other words, on any VM exit or VM entry, the counters should continue counting. This ensures that the counters and the PEBS record information are system-wide.

In this use case, the host owns counter/PEBS configuration, manages the PEBS output buffer, and assures the counter/PEBS state persists across guests, as illustrated by Figure 22-82.

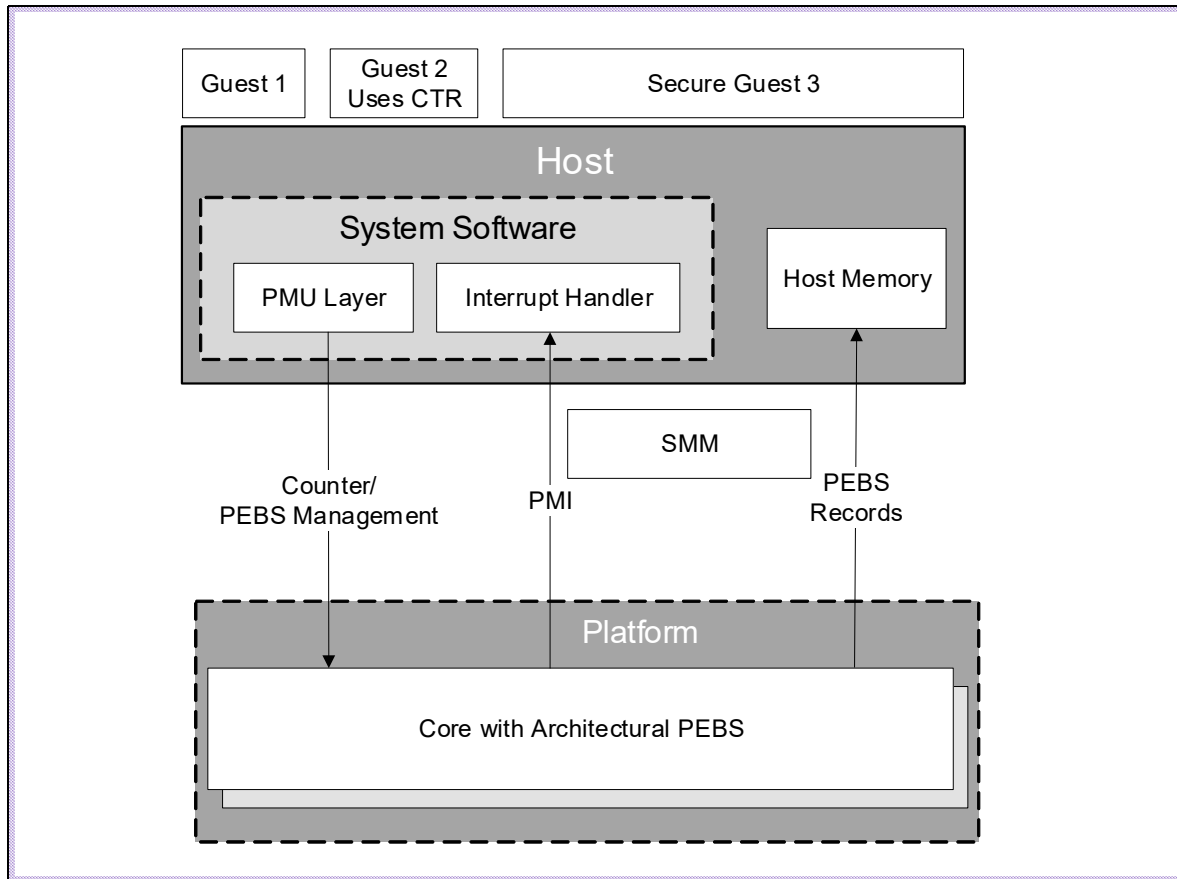


Figure 22-82. System-Wide PEBS: Diagram of Interactions Across Entities

Other use cases, e.g., host-only, are possible but must be managed by the VMM.

22.10.5.2 Intel® Secure Guard Extensions (Intel® SGX)

For production (opt-out) enclaves, an enclave entry freezes counters, except fixed-function counters 1 and 2 (unhalted core cycles and unhalted reference cycles, respectively). Additionally, handling any PEBS event that may occur during an enclave execution is deferred until after the enclave exit, i.e., the PEBS event would skid to the end of EEXIT/AEX. Specifically, the instruction pointer field of the PEBS record would have the EENTER/ERESUME address and the Memory Access Address (MAA) field would be zero.

For non-production (opt-in) enclaves, all counters continue to count. Any PEBS event that may occur during enclave execution would be handled normally. Specifically, fields of the PEBS record would expose the non-production enclave state.

Architectural PEBS will write nothing to memory in the case where the PEBS buffer is mapped to the Processor Reserved Memory Range Register (PRMRR). Writes are directed to the abort page.

22.10.5.3 Intel® Trust Domain Extensions (Intel® TDX)

Intel TDX supports two modes of monitoring:

- TD Profiling – The trusted domain (TD) uses performance-monitoring/PEBS to profile itself.
- System Profiling – The VMM uses performance-monitoring/PEBS to profile itself plus any debuggable TDs.

For TD profiling mode, a TD may use performance-monitoring by setting the TDCS.ATTRIBUTES[PerfMon] bit. This enables a performance-monitoring-enabled TD to access all performance-monitoring MSRs. Architectural PEBS will be enumerated to the TD and the PEBS2GPA flag is set. The TD is expected to set up the PEBS buffer in TD private pages.

Architectural PEBS will not be available for TDs that are not performance-monitoring-enabled. Specifically, Leaf 23H Subleaves 4 and 5 will be cleared, and accesses to the IA32_PEBS_* and IA32_PMC_*_CFG_C MSRs will result in a #GP exception.

22.10.5.4 System Management Mode (SMM)

Performance-monitoring counters count through system management mode (SMM), unless IA32_DE-BUGCTL.FREEZE_WHILE_SMM_EN = 1.

PEBS will be inhibited on opt-out #SMI, inhibit cleared on RSM. This ensures any PEBS pended during SMM will skid to end of RSM and precludes any SMM use of PEBS.

22.10.5.5 SMM-Transfer Monitor (STM)

No STM-specific support is added with Architectural PEBS. However, the STM should configure the SMM-transfer VMCS as follows:

- The “save IA32_PERF_GLOBAL_CTRL” and “load IA32_PERF_GLOBAL_CTRL” VM-exit controls are set to 1.
- The host IA32_PERF_GLOBAL_CTRL value is zero.
- The “load IA32_PERF_GLOBAL_CTRL” VM-entry control is set to 1.

These ensure that non-SMM counters are disabled on an SMM VM exit, that any PEBS pended during such a VM exit are deferred, and that non-SMM counters are re-enabled by a VM entry that returns from SMM.

An STM is free to use PEBS itself, but it should ensure that non-SMM counter configuration state is restored before a VM entry that returns from SMM.

22.11 AUTO COUNTER RELOAD

Auto Counter Reload (ACR) provides a means for software to specify that, for each supported counter, the hardware should automatically reload the counter to a specified initial value upon overflow of chosen counters. This mechanism enables software to sample based on the relative rate of two (or more) events, such that a sample (PMI or PEBS) is taken only if the rate of one event exceeds some threshold relative to the rate of another event. Taking a PMI or PEBS only when the relative rate of performance-monitoring events crosses a threshold can have significantly less performance overhead than other techniques (e.g., taking a PMI every 1000 instructions in order to check the number of mispredicts since the last PMI).

22.11.1 Discovery and Interface

CPUID.23H.02H:EAX indicates general-purpose counters [n:0] that can be reloaded. CPUID.23H.02H:EBX indicates fixed-function counters [m:0] that can be reloaded. CPUID.23H.02H:ECX indicates general-purpose counters [n:0] that can cause a reload of reloadable counters. CPUID.23H.02H:EDX indicates fixed-function counters [m:0] that can cause a reload of reloadable counters. If a counter can be reloaded, its associated reload configuration MSR (*_CFG_B) and its reload value MSR (*_CFG_C) are supported.

See Chapter 2 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for details about the following MSRs: IA32_PMC_GPn_CFG_B, IA32_PMC_GPn_CFG_C, IA32_PMC_FXm_CFG_B, and IA32_PMC_FXm_CFG_C.

22.11.2 Configuration and Behavior

For a given counter IA32_PMC_GPn_CTR, bit fields in the IA32_PMC_GPn_CFG_B MSR indicate which counter(s) can cause a reload of that counter:

- If the general-purpose counter 'n' is configured to do a reload when general-purpose counter 'x' overflows (IA32_PMC_GPn_CFG_B.PMC[x] = 1), then that general-purpose counter 'n' will be written with its reload value (in IA32_PMC_GPn_CFG_C[31:0]) when counter 'x' (IA32_PMC_GPn_CTR) overflows.
- If general-purpose counter 'n' is configured to do a reload when fixed-function counter 'x' overflows (IA32_PMC_GPn_CFG_B.FIXED_CTR[x] = 1), then that general-purpose counter 'n' will be written with its reload value (in IA32_PMC_GPn_CFG_C[31:0]) when fixed-function counter 'x' (IA32_PMC_FXn_CTR) overflows.

ACR will not reload IA32_PMC_GPn_CTR if counters are frozen (IA32_PERF_GLOBAL_STATUS.COUNTERS_FROZEN = 1) or if IA32_PMC_GPn_CTR has already overflowed (IA32_PERF_GLOBAL_STATUS.PMCn_OVF = 1). If a PMI or PEBS is taken due to a counter overflow, the PMI ISR or PEBS record can record the unmodified counter value before reloading the counter. In race conditions, where IA32_PMC_GPn_CTR overflows in the same cycle as a counter configured to reload the IA32_PMC_GPn_CTR on overflow, IA32_PMC_GPn_CTR will not be reloaded, and IA32_PERF_GLOBAL_STATUS.PMCn_OVF will be set.

For counters that reload themselves (i.e., IA32_PMC_GPn_CFG_B.PMCn = 1), the overflow bit (IA32_PERF_GLOBAL_STATUS.PMCn_OVF) will never be set. Instead, upon overflow, the counter will be immediately reloaded; thus, it is never in an overflowed state. There is an exception associated with PEBS; see Section 22.11.2.2.

The behavior is similar for reloading of fixed-function counters. For IA32_PMC_FXm_CTR, the reload value is stored in IA32_PMC_FXm_CFG_C[31:0], and which counters cause reload of IA32_PMC_FXm_CTR is configured in IA32_PMC_FXm_CFG_B.

22.11.2.1 Reload Precision

ACR reload is not guaranteed to be precise; in some cases, a small number of events may be lost during the time between counter overflow and counter reload. However, when the reload happens, hardware will reload all configured counters simultaneously.

22.11.2.2 PEBS Interaction

If a counter is configured to reload other counters with ACR and to take PEBS on overflow, the counter reload actions will be taken only after the PEBS record has been written. This ensures that any counter values captured in the PEBS record reflect the value before the reload occurs. Because the reload actions are taken after the PEBS records are written, reloaded counter value will not account for the events that occurred during the process of writing the PEBS record.

For a counter configured to reload itself and to take PEBS on overflow, the overflow bit associated with the counter (in IA32_PERF_GLOBAL_STATUS) will be set from the time the counter overflows to the time the PEBS record is written. This is required to ensure the PEBS record is not lost due to a VM exit taken during record generation. Once the record is written, the overflow bit will be cleared, and the counter reloaded.

22.11.2.3 Precise Distribution (PDIST) Interaction

Precise distribution of PEBS events (PDIR) is not supported when such a counter is reloaded by ACR. For details on PDIST, see Section 22.9.6.

21. Updates to Chapter 27, Volume 3C

Change bars and text show changes to Chapter 27 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- Added MSRs for FRED in Section 27.4.1, "Guest Register State" and Section 27.5, "Host-State Area".
- Updated descriptions for Bit Positions 0 and 3 in Table 27-5, "Definitions of Pin-Based VM-Execution Controls" (Section 27.6.1).
- Removed reference to the IDT in Section 27.6.3, "Exception Bitmap".
- Updated Table 27-14, "Definitions of Primary VM-Exit Controls" (Section 27.7.1), description for Bit Position by replacing VM-exit interruption-information field with exiting-event identification field.
- Added Save FRED and Load Fred bits to Table 27-15, "Definitions of Secondary VM-Exit Controls".
- Added Load FRED bit to Table 27-17, "Definitions of VM-Entry Controls" (in Section 27.8.1).
- Updated Section 27.8.3, "VM-Entry Controls for Event Injection", for FRED and replacing VM-entry with injected-event and interruption type with event type.
- Added nested exception in Bit Position 13 of Table 27-18, "Format of the Injected-Event Identification Field".
- Updated Section 27.9.2, "Information for VM Exits Due to Vectored Events", for FRED and replacing VM-exit with exiting-event and interruption type with event type.
- Added nested exception in Bit Position 13 of Table 27-20, "Format of the Exiting-Event Identification Field".
- Updated Section 27.9.3, "Information for VM Exits That Occur During Event Delivery", for FRED and replacing IDT-vectoring with original-event and interruption type with event type.
- Added nested exception in Bit Position 13 of Table 27-21, "Format of the Original-Event Identification Field".
- Added Enable PBNDKB (bit 9) to Table 27-8, "Definitions of Tertiary Processor-Based VM-Execution Controls" (in Section 27.6.2).

27.1 OVERVIEW

A logical processor uses **virtual-machine control data structures (VMCSs)** while it is in VMX operation. These manage transitions into and out of VMX non-root operation (VM entries and VM exits) as well as processor behavior in VMX non-root operation. This structure is manipulated by the new instructions VMCLEAR, VMPTRLD, VMREAD, and VMWRITE.

A VMM can use a different VMCS for each virtual machine that it supports. For a virtual machine with multiple logical processors (virtual processors), the VMM can use a different VMCS for each virtual processor.

A logical processor associates a region in memory with each VMCS. This region is called the **VMCS region**.¹ Software references a specific VMCS using the 64-bit physical address of the region (a **VMCS pointer**). VMCS pointers must be aligned on a 4-KByte boundary (bits 11:0 must be zero). These pointers must not set bits beyond the processor's physical-address width.^{2,3}

A logical processor may maintain a number of VMCSs that are **active**. The processor may optimize VMX operation by maintaining the state of an active VMCS in memory, on the processor, or both. At any given time, at most one of the active VMCSs is the **current** VMCS. (This document frequently uses the term "the VMCS" to refer to the current VMCS.) The VMLAUNCH, VMREAD, VMRESUME, and VMWRITE instructions operate only on the current VMCS.

The following items describe how a logical processor determines which VMCSs are active and which is current:

- The memory operand of the VMPTRLD instruction is the address of a VMCS. After execution of the instruction, that VMCS is both active and current on the logical processor. Any other VMCS that had been active remains so, but no other VMCS is current.
- The VMCS link pointer field in the current VMCS (see Section •) is itself the address of a VMCS. If VM entry is performed successfully with the 1-setting of the "VMCS shadowing" VM-execution control, the VMCS referenced by the VMCS link pointer field becomes active on the logical processor. The identity of the current VMCS does not change.
- The memory operand of the VMCLEAR instruction is also the address of a VMCS. After execution of the instruction, that VMCS is neither active nor current on the logical processor. If the VMCS had been current on the logical processor, the logical processor no longer has a current VMCS.

The VMPTRST instruction stores the address of the logical processor's current VMCS into a specified memory location (it stores the value FFFFFFFF_FFFFFFFFH if there is no current VMCS).

The **launch state** of a VMCS determines which VM-entry instruction should be used with that VMCS: the VMLAUNCH instruction requires a VMCS whose launch state is "clear"; the VMRESUME instruction requires a VMCS whose launch state is "launched". A logical processor maintains a VMCS's launch state in the corresponding VMCS region. The following items describe how a logical processor manages the launch state of a VMCS:

- If the launch state of the current VMCS is "clear", successful execution of the VMLAUNCH instruction changes the launch state to "launched".

1. The amount of memory required for a VMCS region is at most 4 KBytes. The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix A.1).

2. CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. This width is used to determine the value of MAXPHYADDR, which constrains VMCS pointers. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

3. If IA32_VMX_BASIC[48] is read as 1, these pointers must not set any bits in the range 63:32; see Appendix A.1.

- The memory operand of the VMCLEAR instruction is the address of a VMCS. After execution of the instruction, the launch state of that VMCS is “clear”.
- There are no other ways to modify the launch state of a VMCS (it cannot be modified using VMWRITE) and there is no direct way to discover it (it cannot be read using VMREAD).

Figure 27-1 illustrates the different states of a VMCS. It uses “X” to refer to the VMCS and “Y” to refer to any other VMCS. Thus: “VMPTRLD X” always makes X current and active; “VMPTRLD Y” always makes X not current (because it makes Y current); VMLAUNCH makes the launch state of X “launched” if X was current and its launch state was “clear”; and VMCLEAR X always makes X inactive and not current and makes its launch state “clear”.

The figure does not illustrate operations that do not modify the VMCS state relative to these parameters (e.g., execution of VMPTRLD X when X is already current). Note that VMCLEAR X makes X “inactive, not current, and clear,” even if X’s current state is not defined (e.g., even if X has not yet been initialized). See Section 27.11.3.

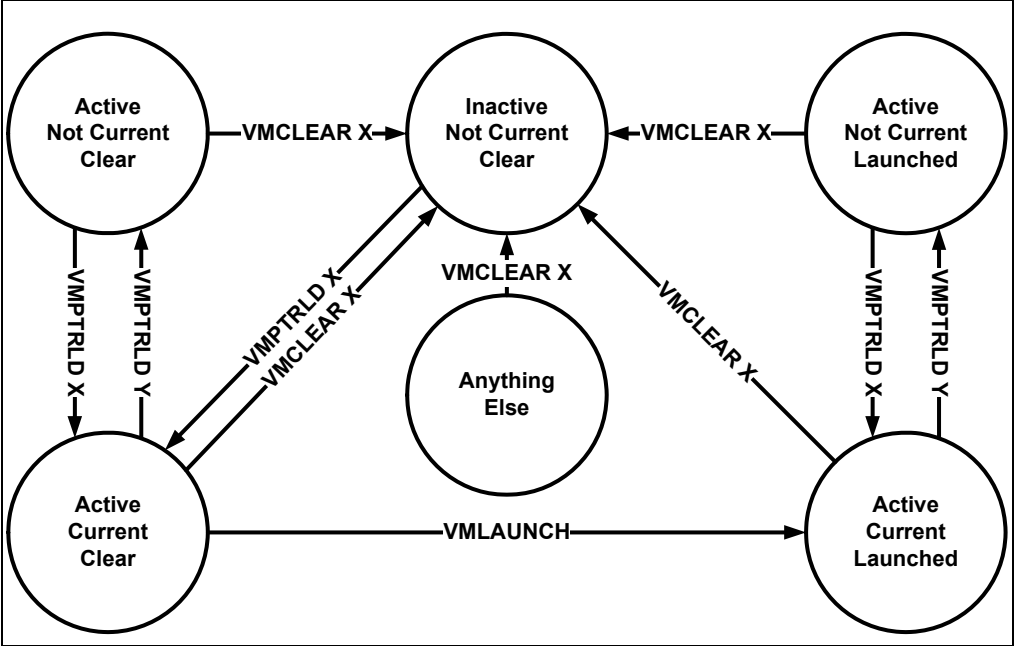


Figure 27-1. States of VMCS X

Because a shadow VMCS (see Section 27.10) cannot be used for VM entry, the launch state of a shadow VMCS is not meaningful. Figure 27-1 does not illustrate all the ways in which a shadow VMCS may be made active.

27.2 FORMAT OF THE VMCS REGION

A VMCS region comprises up to 4-KBytes.¹ The format of a VMCS region is given in Table 27-1.

Table 27-1. Format of the VMCS Region

Byte Offset	Contents
0	Bits 30:0: VMCS revision identifier Bit 31: shadow-VMCS indicator (see Section 27.10)
4	VMX-abort indicator

1. The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix A.1).

Table 27-1. Format of the VMCS Region (Contd.)

Byte Offset	Contents
8	VMCS data (implementation-specific format)

The first 4 bytes of the VMCS region contain the **VMCS revision identifier** at bits 30:0.¹ Processors that maintain VMCS data in different formats (see below) use different VMCS revision identifiers. These identifiers enable software to avoid using a VMCS region formatted for one processor on a processor that uses a different format.² Bit 31 of this 4-byte region indicates whether the VMCS is a shadow VMCS (see Section 27.10).

Software should write the VMCS revision identifier to the VMCS region before using that region for a VMCS. The VMCS revision identifier is never written by the processor; VMPTRLD fails if its operand references a VMCS region whose VMCS revision identifier differs from that used by the processor. (VMPTRLD also fails if the shadow-VMCS indicator is 1 and the processor does not support the 1-setting of the “VMCS shadowing” VM-execution control; see Section 27.6.2) Software can discover the VMCS revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

Software should clear or set the shadow-VMCS indicator depending on whether the VMCS is to be an ordinary VMCS or a shadow VMCS (see Section 27.10). VMPTRLD fails if the shadow-VMCS indicator is set and the processor does not support the 1-setting of the “VMCS shadowing” VM-execution control. Software can discover support for this setting by reading the VMX capability MSR IA32_VMX_PROCBASED_CTL2 (see Appendix A.3.3).

The next 4 bytes of the VMCS region are used for the **VMX-abort indicator**. The contents of these bits do not control processor operation in any way. A logical processor writes a non-zero value into these bits if a VMX abort occurs (see Section 30.7). Software may also write into this field.

The remainder of the VMCS region is used for **VMCS data** (those parts of the VMCS that control VMX non-root operation and the VMX transitions). The format of these data is implementation-specific. VMCS data are discussed in Section 27.3 through Section 27.9. To ensure proper behavior in VMX operation, software should maintain the VMCS region and related structures (enumerated in Section 27.11.4) in writeback cacheable memory. Future implementations may allow or require a different memory type³. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

27.3 ORGANIZATION OF VMCS DATA

The VMCS data are organized into six logical groups:

- **Guest-state area.** Processor state is saved into the guest-state area on VM exits and loaded from there on VM entries.
- **Host-state area.** Processor state is loaded from the host-state area on VM exits.
- **VM-execution control fields.** These fields control processor behavior in VMX non-root operation. They determine in part the causes of VM exits.
- **VM-exit control fields.** These fields control VM exits.
- **VM-entry control fields.** These fields control VM entries.
- **VM-exit information fields.** These fields receive information on VM exits and describe the cause and the nature of VM exits. On some processors, these fields are read-only.⁴

1. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field. For all processors produced prior to this change, bit 31 of the VMCS revision identifier was 0.

2. Logical processors that use the same VMCS revision identifier use the same size for VMCS regions.

3. Alternatively, software may map any of these regions or structures with the UC memory type. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with exceptions noted in Appendix A.1.

4. Software can discover whether these fields can be written by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

The VM-execution control fields, the VM-exit control fields, and the VM-entry control fields are sometimes referred to collectively as VMX controls.

27.4 GUEST-STATE AREA

This section describes fields contained in the guest-state area of the VMCS. VM entries load processor state from these fields and VM exits store processor state into these fields. See Section 29.3.2 and Section 30.3 for details.

27.4.1 Guest Register State

The following fields in the guest-state area correspond to processor registers:

- Control registers CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Debug register DR7 (64 bits; 32 bits on processors that do not support Intel 64 architecture).
- RSP, RIP, and RFLAGS (64 bits each; 32 bits on processors that do not support Intel 64 architecture).¹
- The following fields for each of the registers CS, SS, DS, ES, FS, GS, LDTR, and TR:
 - Selector (16 bits).
 - Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture). The base-address fields for CS, SS, DS, and ES have only 32 architecturally-defined bits; nevertheless, the corresponding VMCS fields have 64 bits on processors that support Intel 64 architecture.
 - Segment limit (32 bits). The limit field is always a measure in bytes.
 - Access rights (32 bits). The format of this field is given in Table 27-2 and detailed as follows:
 - The low 16 bits correspond to bits 23:8 of the upper 32 bits of a 64-bit segment descriptor. While bits 19:16 of code-segment and data-segment descriptors correspond to the upper 4 bits of the segment limit, the corresponding bits (bits 11:8) are reserved in this VMCS field.
 - Bit 16 indicates an **unusable segment**. Attempts to use such a segment fault except in 64-bit mode. In general, a segment register is unusable if it has been loaded with a null selector.²
 - Bits 31:17 are reserved.

Table 27-2. Format of Access Rights

Bit Position(s)	Field
3:0	Segment type
4	S — Descriptor type (0 = system; 1 = code or data)
6:5	DPL — Descriptor privilege level
7	P — Segment present

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.
2. There are a few exceptions to this statement. For example, a segment with a non-null selector may be unusable following a task switch that fails after its commit point; see “Interrupt 10—Invalid TSS Exception (#TS)” in Section 7.14, “Exception and Interrupt Handling in 64-bit Mode,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. In contrast, the TR register is usable after processor reset despite having a null selector; see Table 13-1 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Table 27-2. Format of Access Rights (Contd.)

Bit Position(s)	Field
11:8	Reserved
12	AVL — Available for use by system software
13	Reserved (except for CS) L — 64-bit mode active (for CS only)
14	D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
15	G — Granularity
16	Segment unusable (0 = usable; 1 = unusable)
31:17	Reserved

The base address, segment limit, and access rights compose the “hidden” part (or “descriptor cache”) of each segment register. These data are included in the VMCS because it is possible for a segment register’s descriptor cache to be inconsistent with the segment descriptor in memory (in the GDT or the LDT) referenced by the segment register’s selector.

The value of the DPL field for SS is always equal to the logical processor’s current privilege level (CPL).¹

On some processors, executions of VMWRITE ignore attempts to write non-zero values to any of bits 11:8 or bits 31:17. On such processors, VMREAD always returns 0 for those bits, and VM entry treats those bits as if they were all 0 (see Section 29.3.1.2).

- The following fields for each of the registers GDTR and IDTR:
 - Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture).
 - Limit (32 bits). The limit fields contain 32 bits even though these fields are specified as only 16 bits in the architecture.
- The following MSRs:
 - IA32_DEBUGCTL (64 bits)
 - IA32_SYSENTER_CS (32 bits)
 - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture)
 - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-entry control.
 - IA32_PAT (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_PAT” VM-entry control or that of the “save IA32_PAT” VM-exit control.
 - IA32_EFER (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_EFER” VM-entry control or that of the “save IA32_EFER” VM-exit control.
 - IA32_BNDCFGS (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_BNDCFGS” VM-entry control or that of the “clear IA32_BNDCFGS” VM-exit control.
 - IA32_RTIT_CTL (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_RTIT_CTL” VM-entry control or that of the “clear IA32_RTIT_CTL” VM-exit control.
 - IA32_LBR_CTL (64 bits). This field is supported only on processors that support either the 1-setting of the “load guest IA32_LBR_CTL” VM-entry control or that of the “clear IA32_LBR_CTL” VM-exit control.
 - IA32_S_CET (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.

1. In protected mode, CPL is also associated with the RPL field in the CS selector. However, the RPL fields are not meaningful in real-address mode or in virtual-8086 mode.

- IA32_INTERRUPT_SSP_TABLE_ADDR (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.
- IA32_PKRS (64 bits). This field is supported only on processors that support the 1-setting of the “load PKRS” VM-entry control.
- The following MSRs on processors that support either the 1-setting of the “load FRED” VM-entry control or that of the “save FRED” VM-exit control (each is 64 bits):
 - IA32_FRED_CONFIG
 - IA32_FRED_RSP1
 - IA32_FRED_RSP2
 - IA32_FRED_RSP3
 - IA32_FRED_STKLVL5
 - IA32_FRED_SSP1
 - IA32_FRED_SSP2
 - IA32_FRED_SSP3
- The shadow-stack pointer register SSP (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.
- The register SMBASE (32 bits). This register contains the base address of the logical processor’s SMRAM image.

27.4.2 Guest Non-Register State

In addition to the register state described in Section 27.4.1, the guest-state area includes the following fields that characterize guest state but which do not correspond to processor registers:

- **Activity state** (32 bits). This field identifies the logical processor’s activity state. When a logical processor is executing instructions normally, it is in the **active state**. Execution of certain instructions and the occurrence of certain events may cause a logical processor to transition to an **inactive state** in which it ceases to execute instructions.

The following activity states are defined:¹

- 0: **Active**. The logical processor is executing instructions normally.
- 1: **HLT**. The logical processor is inactive because it executed the HLT instruction.
- 2: **Shutdown**. The logical processor is inactive because it incurred a **triple fault**² or some other serious error.
- 3: **Wait-for-SIPI**. The logical processor is inactive because it is waiting for a startup-IPI (SIPI).

Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine what activity states are supported.

- **Interruptibility state** (32 bits). The IA-32 architecture includes features that permit certain events to be blocked for a period of time. This field contains information about such blocking. Details and the format of this field are given in Table 27-3.
- **Pending debug exceptions** (64 bits; 32 bits on processors that do not support Intel 64 architecture). IA-32 processors may recognize one or more debug exceptions without immediately delivering them.³ This field contains information about such exceptions. This field is described in Table 27-4.

1. Execution of the MWAIT instruction may put a logical processor into an inactive state. However, this VMCS field never reflects this state. See Section 30.1.

2. A triple fault occurs when a logical processor encounters an exception while attempting to deliver a double fault.

Table 27-3. Format of Interruptibility State

Bit Position(s)	Bit Name	Notes
0	Blocking by STI	See the “STI—Set Interrupt Flag” section in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B. Execution of STI with RFLAGS.IF = 0 blocks maskable interrupts on the instruction boundary following its execution. ¹ Setting this bit indicates that this blocking is in effect.
1	Blocking by MOV SS	See Section 7.8.3, “Masking Exceptions and Interrupts When Switching Stacks,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. Execution of a MOV to SS or a POP to SS blocks or suppresses certain debug exceptions as well as interrupts (maskable and nonmaskable) on the instruction boundary following its execution. Setting this bit indicates that this blocking is in effect. ² This document uses the term “blocking by MOV SS,” but it applies equally to POP SS.
2	Blocking by SMI	See Section 34.2, “System Management Interrupt (SMI).” System-management interrupts (SMIs) are disabled while the processor is in system-management mode (SMM). Setting this bit indicates that blocking of SMIs is in effect.
3	Blocking by NMI	See Section 7.7.1, “Handling Multiple NMIs,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A and Section 34.8, “NMI Handling While in SMM.” Delivery of a non-maskable interrupt (NMI) or a system-management interrupt (SMI) blocks subsequent NMIs until the next execution of IRET. See Section 28.3 for how this behavior of IRET may change in VMX non-root operation. Setting this bit indicates that blocking of NMIs is in effect. Clearing this bit does not imply that NMIs are not (temporarily) blocked for other reasons. If the “virtual NMIs” VM-execution control (see Section 27.6.1) is 1, this bit does not control the blocking of NMIs. Instead, it refers to “virtual-NMI blocking” (the fact that guest software is not ready for an NMI).
4	Enclave interruption	Set to 1 if the VM exit occurred while the logical processor was in enclave mode. Such VM exits includes those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode. A VM exit also sets this bit if it was due to or incident to delivery of an event that was either (1) made pending or injected by VM entry while the guest interruptibility-state field sets this bit; or (2) pending following an execution of RSM that occurred while bit 1 was set in the byte at offset 7EE0H in the state save image in SMRAM.
31:5	Reserved	VM entry will fail if these bits are not 0. See Section 29.3.1.5.

NOTES:

1. Nonmaskable interrupts and system-management interrupts may also be inhibited on the instruction boundary following such an execution of STI.
 2. System-management interrupts may also be inhibited on the instruction boundary following such an execution of MOV or POP.
- **VMCS link pointer** (64 bits). If the “VMCS shadowing” VM-execution control is 1, the VMREAD and VMWRITE instructions access the VMCS referenced by this pointer (see Section 27.10). Otherwise, software should set this field to FFFFFFFF_FFFFFFFFH to avoid VM-entry failures (see Section 29.3.1.5).
-
3. For example, execution of a MOV to SS or a POP to SS may inhibit some debug exceptions for one instruction. See Section 7.8.3 of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. In addition, certain events incident to an instruction (for example, an INIT signal) may take priority over debug traps generated by that instruction. See Table 7-2 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Table 27-4. Format of Pending-Debug-Exceptions

Bit Position(s)	Bit Name	Notes
3:0	B3 – B0	When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if the corresponding enabling bit in DR7 is not set.
10:4	Reserved	VM entry fails if these bits are not 0. See Section 29.3.1.5.
11	BLD	When set, this bit indicates that a bus lock was asserted while OS bus-lock detection was enabled and CPL > 0 (see Section 20.3.1.6, “OS Bus-Lock Detection”). ¹
12	Enabled breakpoint	When set, this bit indicates that at least one data or I/O breakpoint was met and was enabled in DR7; the XBEGIN instruction was executed immediately before the VM exit and advanced debugging of RTM transactional regions had been enabled; or a bus lock was asserted while CPL > 0 and OS bus-lock detection had been enabled.
13	Reserved	VM entry fails if this bit is not 0. See Section 29.3.1.5.
14	BS	When set, this bit indicates that a debug exception would have been triggered by single-step execution mode.
15	Reserved	VM entry fails if this bit is not 0. See Section 29.3.1.5.
16	RTM	When set, this bit indicates that a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). ²
63:17	Reserved	VM entry fails if these bits are not 0. See Section 29.3.1.5. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 11 to indicate detection of a bus lock, while this field **sets** the bit to indicate that condition.
2. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 16 to indicate an RTM-related exception, while this field **sets** the bit to indicate that condition.

- **VMX-preemption timer value** (32 bits). This field is supported only on processors that support the 1-setting of the “activate VMX-preemption timer” VM-execution control. This field contains the value that the VMX-preemption timer will use following the next VM entry with that setting. See Section 28.5.1 and Section 29.7.4.
- **Page-directory-pointer-table entries** (PDPTes; 64 bits each). These four (4) fields (PDPTe0, PDPTe1, PDPTe2, and PDPTe3) are supported only on processors that support the 1-setting of the “enable EPT” VM-execution control. They correspond to the PDPTes referenced by CR3 when PAE paging is in use (see Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). They are used only if the “enable EPT” VM-execution control is 1.
- **Guest interrupt status** (16 bits). This field is supported only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control. It characterizes part of the guest’s virtual-APIC state and does not correspond to any processor or APIC registers. It comprises two 8-bit subfields:
 - **Requesting virtual interrupt (RVI)**. This is the low byte of the guest interrupt status. The processor treats this value as the vector of the highest priority virtual interrupt that is requesting service. (The value 0 implies that there is no such interrupt.)
 - **Servicing virtual interrupt (SVI)**. This is the high byte of the guest interrupt status. The processor treats this value as the vector of the highest priority virtual interrupt that is in service. (The value 0 implies that there is no such interrupt.)

See Chapter 32 for more information on the use of this field.
- **PML index** (16 bits). This field is supported only on processors that support the 1-setting of the “enable PML” VM-execution control. It contains the logical index of the next entry in the page-modification log. Because the

page-modification log comprises 512 entries, the PML index is typically a value in the range 0–511. Details of the page-modification log and use of the PML index are given in Section 31.3.6.

27.5 HOST-STATE AREA

This section describes fields contained in the host-state area of the VMCS. As noted earlier, processor state is loaded from these fields on every VM exit (see Section 30.5).

All fields in the host-state area correspond to processor registers:

- CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- RSP and RIP (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Selector fields (16 bits each) for the segment registers CS, SS, DS, ES, FS, GS, and TR. There is no field in the host-state area for the LDTR selector.
- Base-address fields for FS, GS, TR, GDTR, and IDTR (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- The following MSRs:
 - IA32_SYSENTER_CS (32 bits)
 - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture).
 - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-exit control.
 - IA32_PAT (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PAT” VM-exit control.
 - IA32_EFER (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_EFER” VM-exit control.
 - IA32_S_CET (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.
 - IA32_INTERRUPT_SSP_TABLE_ADDR (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.
 - IA32_PKRS (64 bits). This field is supported only on processors that support the 1-setting of the “load PKRS” VM-exit control.
 - The following MSRs on processors that support the 1-setting of the “load FRED” VM-exit control (each is 64 bits):
 - IA32_FRED_CONFIG
 - IA32_FRED_RSP1
 - IA32_FRED_RSP2
 - IA32_FRED_RSP3
 - IA32_FRED_STKLVLS
 - IA32_FRED_SSP1
 - IA32_FRED_SSP2
 - IA32_FRED_SSP3
- The shadow-stack pointer register SSP (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.

In addition to the state identified here, some processor state components are loaded with fixed values on every VM exit; there are no fields corresponding to these components in the host-state area. See Section 30.5 for details of how state is loaded on VM exits.

27.6 VM-EXECUTION CONTROL FIELDS

The VM-execution control fields govern VMX non-root operation. These are described in Section 27.6.1 through Section 27.6.8.

27.6.1 Pin-Based VM-Execution Controls

The pin-based VM-execution controls constitute a 32-bit vector that governs the handling of asynchronous events (for example: interrupts).¹ Table 27-5 lists the controls. See Chapter 29 for how these controls affect processor behavior in VMX non-root operation.

Table 27-5. Definitions of Pin-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	External-interrupt exiting	If this control is 1, external interrupts cause VM exits. Otherwise, they are delivered normally. If this control is 1, the value of RFLAGS.IF does not affect interrupt blocking.
3	NMI exiting	If this control is 1, non-maskable interrupts (NMIs) cause VM exits. Otherwise, they are delivered normally using vector 2 . This control also determines interactions between IRET and blocking by NMI (see Section 28.3).
5	Virtual NMIs	If this control is 1, NMIs are never blocked and the “blocking by NMI” bit (bit 3) in the interruptibility-state field indicates “virtual-NMI blocking” (see Table 27-3). This control also interacts with the “NMI-window exiting” VM-execution control (see Section 27.6.2).
6	Activate VMX-preemption timer	If this control is 1, the VMX-preemption timer counts down in VMX non-root operation; see Section 28.5.1. A VM exit occurs when the timer counts down to zero; see Section 28.2.
7	Process posted interrupts	If this control is 1, the processor treats interrupts with the posted-interrupt notification vector (see Section 27.6.8) specially, updating the virtual-APIC page with posted-interrupt requests (see Section 32.6).

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs `IA32_VMX_PINBASED_CTLS` and `IA32_VMX_TRUE_PINBASED_CTLS` (see Appendix A.3.1) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 29.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 2, and 4. The VMX capability MSR `IA32_VMX_PINBASED_CTLS` will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR `IA32_VMX_TRUE_PINBASED_CTLS` MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

27.6.2 Processor-Based VM-Execution Controls

The processor-based VM-execution controls constitute three vectors that govern the handling of synchronous events, mainly those caused by the execution of specific instructions.² These are the **primary processor-based VM-execution controls** (32 bits), the **secondary processor-based VM-execution controls** (32 bits), and the tertiary **VM-execution controls** (64 bits).

1. Some asynchronous events cause VM exits regardless of the settings of the pin-based VM-execution controls (see Section 28.2).
2. Some instructions cause VM exits regardless of the settings of the processor-based VM-execution controls (see Section 28.1.2), as do task switches (see Section 28.2).

Table 27-6 lists the primary processor-based VM-execution controls. See Chapter 27 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 27-6. Definitions of Primary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
2	Interrupt-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if RFLAGS.IF = 1 and there are no other blocking of interrupts (see Section 27.6.1).
3	Use TSC offsetting	This control determines whether executions of RDTSC, executions of RDTSCP, and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC offset field (see Section 27.6.5 and Section 28.3).
7	HLT exiting	This control determines whether executions of HLT cause VM exits.
9	INVLPG exiting	This determines whether executions of INVLPG and INVPCID cause VM exits.
10	MWAIT exiting	This control determines whether executions of MWAIT cause VM exits.
11	RDPMC exiting	This control determines whether executions of RDPMC cause VM exits.
12	RDTSC exiting	This control determines whether executions of RDTSC and RDTSCP cause VM exits.
15	CR3-load exiting	In conjunction with the CR3-target controls (see Section 27.6.7), this control determines whether executions of MOV to CR3 cause VM exits. See Section 28.1.3. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
16	CR3-store exiting	This control determines whether executions of MOV from CR3 cause VM exits. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
17	Activate tertiary controls	This control determines whether the tertiary processor-based VM-execution controls are used. If this control is 0, the logical processor operates as if all the tertiary processor-based VM-execution controls were also 0.
19	CR8-load exiting	This control determines whether executions of MOV to CR8 cause VM exits.
20	CR8-store exiting	This control determines whether executions of MOV from CR8 cause VM exits.
21	Use TPR shadow	Setting this control to 1 enables TPR virtualization and other APIC-virtualization features. See Chapter 32.
22	NMI-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if there is no virtual-NMI blocking (see Section 27.6.1).
23	MOV-DR exiting	This control determines whether executions of MOV DR cause VM exits.
24	Unconditional I/O exiting	This control determines whether executions of I/O instructions (IN, INS/INSB/INSW/INSD, OUT, and OUTS/OUTSB/OUTSW/OUTSD) cause VM exits.
25	Use I/O bitmaps	This control determines whether I/O bitmaps are used to restrict executions of I/O instructions (see Section 27.6.4 and Section 28.1.3). For this control, “0” means “do not use I/O bitmaps” and “1” means “use I/O bitmaps.” If the I/O bitmaps are used, the setting of the “unconditional I/O exiting” control is ignored.
27	Monitor trap flag	If this control is 1, the monitor trap flag debugging feature is enabled. See Section 28.5.2.
28	Use MSR bitmaps	This control determines whether MSR bitmaps are used to control execution of the RDMSR and WRMSR instructions (see Section 27.6.9 and Section 28.1.3). For this control, “0” means “do not use MSR bitmaps” and “1” means “use MSR bitmaps.” If the MSR bitmaps are not used, all executions of the RDMSR and WRMSR instructions cause VM exits.
29	MONITOR exiting	This control determines whether executions of MONITOR cause VM exits.
30	PAUSE exiting	This control determines whether executions of PAUSE cause VM exits.
31	Activate secondary controls	This control determines whether the secondary processor-based VM-execution controls are used. If this control is 0, the logical processor operates as if all the secondary processor-based VM-execution controls were also 0.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_PROCBASED_CTLS and IA32_VMX_TRUE_PROCBASED_CTLS (see Appendix A.3.2) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 29.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 4–6, 8, 13–16, and 26. The VMX capability MSR IA32_VMX_PROCBASED_CTLS will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_PROCBASED_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-execution controls determines whether the secondary processor-based VM-execution controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the secondary processor-based VM-execution controls were 0. Processors that support only the 0-setting of bit 31 of the primary processor-based VM-execution controls do not support the secondary processor-based VM-execution controls.

Table 27-7 lists the secondary processor-based VM-execution controls. See Chapter 27 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 27-7. Definitions of Secondary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	Virtualize APIC accesses	If this control is 1, the logical processor treats specially accesses to the page with the APIC-access address. See Section 32.4.
1	Enable EPT	If this control is 1, extended page tables (EPT) are enabled. See Section 31.3.
2	Descriptor-table exiting	This control determines whether executions of LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, and STR cause VM exits.
3	Enable RDTSCP	If this control is 0, any execution of RDTSCP causes an invalid-opcode exception (#UD).
4	Virtualize x2APIC mode	If this control is 1, the logical processor treats specially RDMSR and WRMSR to APIC MSRs (in the range 800H–8FFH). See Section 32.5.
5	Enable VPID	If this control is 1, cached translations of linear addresses are associated with a virtual-processor identifier (VPID). See Section 31.1.
6	WBINVD exiting	This control determines whether executions of WBINVD and WBNOINVD cause VM exits.
7	Unrestricted guest	This control determines whether guest software may run in unpaged protected mode or in real-address mode.
8	APIC-register virtualization	If this control is 1, the logical processor virtualizes certain APIC accesses. See Section 32.4 and Section 32.5.
9	Virtual-interrupt delivery	This controls enables the evaluation and delivery of pending virtual interrupts as well as the emulation of writes to the APIC registers that control interrupt prioritization.
10	PAUSE-loop exiting	This control determines whether a series of executions of PAUSE can cause a VM exit (see Section 27.6.13 and Section 28.1.3).
11	RDRAND exiting	This control determines whether executions of RDRAND cause VM exits.
12	Enable INVPCID	If this control is 0, any execution of INVPCID causes a #UD.
13	Enable VM functions	Setting this control to 1 enables use of the VMFUNC instruction in VMX non-root operation. See Section 28.5.6.
14	VMCS shadowing	If this control is 1, executions of VMREAD and VMWRITE in VMX non-root operation may access a shadow VMCS (instead of causing VM exits). See Section 27.10 and Section 33.3.
15	Enable ENCLS exiting	If this control is 1, executions of ENCLS consult the ENCLS-exiting bitmap to determine whether the instruction causes a VM exit. See Section 27.6.16 and Section 28.1.3.
16	RDSEED exiting	This control determines whether executions of RDSEED cause VM exits.
17	Enable PML	If this control is 1, an access to a guest-physical address that sets an EPT dirty bit first adds an entry to the page-modification log. See Section 31.3.6.

Table 27-7. Definitions of Secondary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
18	EPT-violation #VE	If this control is 1, EPT violations may cause virtualization exceptions (#VE) instead of VM exits. See Section 28.5.7.
19	Conceal VMX from PT	If this control is 1, Intel Processor Trace suppresses from PIPs an indication that the processor was in VMX non-root operation and omits a VMCS packet from any PSB+ produced in VMX non-root operation (see Chapter 36).
20	Enable XSAVES/XRSTORS	If this control is 0, any execution of XSAVES or XRSTORS causes a #UD.
21	PASID translation	If this control is 1, PASID translation is performed for executions of ENQCMD and ENQCMLS. See Section 28.5.8.
22	Mode-based execute control for EPT	If this control is 1, EPT execute permissions are based on whether the linear address being accessed is supervisor mode or user mode. See Chapter 31.
23	Sub-page write permissions for EPT	If this control is 1, EPT write permissions may be specified at the granularity of 128 bytes. See Section 31.3.4.
24	Intel PT uses guest physical addresses	If this control is 1, all output addresses used by Intel Processor Trace are treated as guest-physical addresses and translated using EPT. See Section 28.5.4.
25	Use TSC scaling	This control determines whether executions of RDTSC, executions of RDTSCP, and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC multiplier field (see Section 27.6.5 and Section 28.3).
26	Enable user wait and pause	If this control is 0, any execution of TPAUSE, UMONITOR, or UMWAIT causes a #UD.
27	Enable PCONFIG	If this control is 0, any execution of PCONFIG causes a #UD.
30	VMM bus-lock detection	This control determines whether assertion of a bus lock causes a VM exit. See Section 28.2.
31	Instruction timeout	If this control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within a specified amount of time. See Section 27.6.24 and Section 28.2.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROC-BASED_CTLSS2 (see Appendix A.3.3) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 29.2.1.1).

Bit 17 of the primary processor-based VM-execution controls determines whether the tertiary processor-based VM-execution controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the tertiary processor-based VM-execution controls were 0. Processors that support only the 0-setting of bit 17 of the primary processor-based VM-execution controls do not support the tertiary processor-based VM-execution controls.

Table 27-8 lists the tertiary processor-based VM-execution controls. See Chapter 27 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 27-8. Definitions of Tertiary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	LOADIWKEY exiting	This control determines whether executions of LOADIWKEY cause VM exits.
1	Enable HLAT	This control enables hypervisor-managed linear-address translation. See Section 5.5.1.
2	EPT paging-write control	If this control is 1, EPT permissions can be specified to allow writes only for paging-related updates. See Section 31.3.3.2.
3	Guest-paging verification	If this control is 1, EPT permissions can be specified to prevent accesses using linear addresses whose translation has certain properties. See Section 31.3.3.2.
4	IPI virtualization	If this control is 1, virtualization of interprocessor interrupts (IPIs) is enabled. See Section 32.1.6.

Table 27-8. Definitions of Tertiary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
5	SEAM guest-physical address width	This control determines the operation of EPT in SEAM non-root operation. See Section 35.3.2.
6	Enable MSR-list instructions	If this control is 0, any execution of RDMSRLIST or WRMSRLIST causes a #UD.
7	Virtualize IA32_SPEC_CTRL	If this control is 1, the operation of the RDMSR and WRMSR instructions is changed when accessing the IA32_SPEC_CTRL MSR. See Section 26.3.
9	Enable PBNDKB	If this control is 0, any execution of the PBNDKB causes a #UD.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROCBASED_CTL3 (see Appendix A.3.4) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 29.2.1.1).

27.6.3 Exception Bitmap

The **exception bitmap** is a 32-bit field that contains one bit for each exception. When an exception occurs, its vector is used to select a bit in this field. If the bit is 1, the exception causes a VM exit. If the bit is 0, the exception is delivered normally using the exception's vector.

Whether a page fault (exception with vector 14) causes a VM exit is determined by bit 14 in the exception bitmap as well as the error code produced by the page fault and two 32-bit fields in the VMCS (the **page-fault error-code mask** and **page-fault error-code match**). See Section 28.2 for details.

27.6.4 I/O-Bitmap Addresses

The VM-execution control fields include the 64-bit physical addresses of **I/O bitmaps** A and B (each of which are 4 KBytes in size). I/O bitmap A contains one bit for each I/O port in the range 0000H through 7FFFH; I/O bitmap B contains bits for ports in the range 8000H through FFFFH.

A logical processor uses these bitmaps if and only if the "use I/O bitmaps" control is 1. If the bitmaps are used, execution of an I/O instruction causes a VM exit if any bit in the I/O bitmaps corresponding to a port it accesses is 1. See Section 28.1.3 for details. If the bitmaps are used, their addresses must be 4-KByte aligned.

27.6.5 Time-Stamp Counter Offset and Multiplier

The VM-execution control fields include a 64-bit **TSC-offset** field. If the "RDTSC exiting" control is 0 and the "use TSC offsetting" control is 1, this field controls executions of the RDTSC and RDTSCP instructions. It also controls executions of the RDMSR instruction that read from the IA32_TIME_STAMP_COUNTER MSR. For all of these, the value of the TSC offset is added to the value of the time-stamp counter, and the sum is returned to guest software in EDX:EAX.

Processors that support the 1-setting of the "use TSC scaling" control also support a 64-bit **TSC-multiplier** field. If this control is 1 (and the "RDTSC exiting" control is 0 and the "use TSC offsetting" control is 1), this field also affects the executions of the RDTSC, RDTSCP, and RDMSR instructions identified above. Specifically, the contents of the time-stamp counter is first multiplied by the TSC multiplier before adding the TSC offset.

See Chapter 27 for a detailed treatment of the behavior of RDTSC, RDTSCP, and RDMSR in VMX non-root operation.

27.6.6 Guest/Host Masks and Read Shadows for CR0 and CR4

VM-execution control fields include **guest/host masks** and **read shadows** for the CR0 and CR4 registers. These fields control executions of instructions that access those registers (including CLTS, LMSW, MOV CR, and SMSW). They are 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

In general, bits set to 1 in a guest/host mask correspond to bits “owned” by the host:

- Guest attempts to set them (using CLTS, LMSW, or MOV to CR) to values differing from the corresponding bits in the corresponding read shadow cause VM exits.
- Guest reads (using MOV from CR or SMSW) return values for these bits from the corresponding read shadow.

Bits cleared to 0 correspond to bits “owned” by the guest; guest attempts to modify them succeed and guest reads return values for these bits from the control register itself.

See Chapter 29 for details regarding how these fields affect VMX non-root operation.

27.6.7 CR3-Target Controls

The VM-execution control fields include a set of 4 **CR3-target values** and a **CR3-target count**. The CR3-target values each have 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not. The CR3-target count has 32 bits on all processors.

An execution of MOV to CR3 in VMX non-root operation does not cause a VM exit if its source operand matches one of these values. If the CR3-target count is n , only the first n CR3-target values are considered; if the CR3-target count is 0, MOV to CR3 always causes a VM exit.

There are no limitations on the values that can be written for the CR3-target values. VM entry fails (see Section 29.2) if the CR3-target count is greater than 4.

Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine the number of values supported.

27.6.8 Controls for APIC Virtualization

There are three mechanisms by which software accesses registers of the logical processor’s local APIC:

- If the local APIC is in xAPIC mode, it can perform memory-mapped accesses to addresses in the 4-KByte page referenced by the physical address in the IA32_APIC_BASE MSR (see Section 13.4.4, “Local APIC Status and Location,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, and the Intel® 64 Architecture Processor Topology Enumeration Technical Paper).¹
- If the local APIC is in x2APIC mode, it can access the local APIC’s registers using the RDMSR and WRMSR instructions (see the Intel® 64 Architecture Processor Topology Enumeration Technical Paper).
- In 64-bit mode, it can access the local APIC’s task-priority register (TPR) using the MOV CR8 instruction.

Several processor-based VM-execution controls (see Section 27.6.2) control such accesses. These are “use TPR shadow”, “virtualize APIC accesses”, “virtualize x2APIC mode”, “virtual-interrupt delivery”, “APIC-register virtualization”, and “IPI virtualization”. These controls interact with the following fields:

- **APIC-access address** (64 bits). This field contains the physical address of the 4-KByte **APIC-access page**. If the “virtualize APIC accesses” VM-execution control is 1, access to this page may cause VM exits or be virtualized by the processor. See Section 32.4.

The APIC-access address exists only on processors that support the 1-setting of the “virtualize APIC accesses” VM-execution control.

- **Virtual-APIC address** (64 bits). This field contains the physical address of the 4-KByte **virtual-APIC page**. The processor uses the virtual-APIC page to virtualize certain accesses to APIC registers and to manage virtual interrupts; see Chapter 32.

Depending on the setting of the controls indicated earlier, the virtual-APIC page may be accessed by the following operations:

- The MOV CR8 instructions (see Section 32.3).
- Accesses to the APIC-access page if, in addition, the “virtualize APIC accesses” VM-execution control is 1 (see Section 32.4).

1. If the local APIC does not support x2APIC mode, it is always in xAPIC mode.

- The RDMSR and WRMSR instructions if, in addition, the value of ECX is in the range 800H–8FFH (indicating an APIC MSR) and the “virtualize x2APIC mode” VM-execution control is 1 (see Section 32.5).

If the “use TPR shadow” VM-execution control is 1, VM entry ensures that the virtual-APIC address is 4-KByte aligned. The virtual-APIC address exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.

- **TPR threshold** (32 bits). Bits 3:0 of this field determine the threshold below which bits 7:4 of VTPR (see Section 32.1.1) cannot fall. If the “virtual-interrupt delivery” VM-execution control is 0, a VM exit occurs after an operation (e.g., an execution of MOV to CR8) that reduces the value of those bits below the TPR threshold. See Section 32.1.2.

The TPR threshold exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.

- **EOI-exit bitmap** (4 fields; 64 bits each). These fields are supported only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control. They are used to determine which virtualized writes to the APIC’s EOI register cause VM exits:
 - EOI_EXIT0 contains bits for vectors from 0 (bit 0) to 63 (bit 63).
 - EOI_EXIT1 contains bits for vectors from 64 (bit 0) to 127 (bit 63).
 - EOI_EXIT2 contains bits for vectors from 128 (bit 0) to 191 (bit 63).
 - EOI_EXIT3 contains bits for vectors from 192 (bit 0) to 255 (bit 63).

See Section 32.1.4 for more information on the use of this field.

- **Posted-interrupt notification vector** (16 bits). This field is supported only on processors that support the 1-setting of the “process posted interrupts” VM-execution control. Its low 8 bits contain the interrupt vector that is used to notify a logical processor that virtual interrupts have been posted. See Section 32.6 for more information on the use of this field.
- **Posted-interrupt descriptor address** (64 bits). This field is supported only on processors that support the 1-setting of the “process posted interrupts” VM-execution control. It is the physical address of a 64-byte aligned posted interrupt descriptor. See Section 32.6 for more information on the use of this field.
- **PID-pointer table address** (64 bits). This field contains the physical address of the **PID-pointer table**. If the “IPI virtualization” VM-execution control is 1, the logical processor uses entries in this table to virtualize IPIs. See Section 32.1.6.
- **Last PID-pointer index** (16 bits). This field contains the index of the last entry in the PID-pointer table.

27.6.9 MSR-Bitmap Address

On processors that support the 1-setting of the “use MSR bitmaps” VM-execution control, the VM-execution control fields include the 64-bit physical address of four contiguous **MSR bitmaps**, which are each 1-KByte in size. This field does not exist on processors that do not support the 1-setting of that control. The four bitmaps are:

- **Read bitmap for low MSRs** (located at the MSR-bitmap address). This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Read bitmap for high MSRs** (located at the MSR-bitmap address plus 1024). This contains one bit for each MSR address in the range C0000000H to C0001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Write bitmap for low MSRs** (located at the MSR-bitmap address plus 2048). This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.
- **Write bitmap for high MSRs** (located at the MSR-bitmap address plus 3072). This contains one bit for each MSR address in the range C0000000H to C0001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.

A logical processor uses these bitmaps if and only if the “use MSR bitmaps” control is 1. If the bitmaps are used, an execution of RDMSR or WRMSR causes a VM exit if the value of RCX is in neither of the ranges covered by the

bitmaps or if the appropriate bit in the MSR bitmaps (corresponding to the instruction and the RCX value) is 1. See Section 28.1.3 for details. If the bitmaps are used, their address must be 4-KByte aligned.

27.6.10 Executive-VMCS Pointer

The executive-VMCS pointer is a 64-bit field used in the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). SMM VM exits save this field as described in Section 34.15.2. VM entries that return from SMM use this field as described in Section 34.15.4.

27.6.11 Extended-Page-Table Pointer (EPTP)

The **extended-page-table pointer** (EPTP) contains the address of EPT paging structure that is the root of the hierarchy of structures that control the translation of guest-physical addresses (see Section 31.3.2), as well as other EPT configuration information. The format of this field is shown in Table 27-9.

Table 27-9. Format of Extended-Page-Table Pointer

Bit Position(s)	Field
2:0	EPT paging-structure memory type (see Section 31.3.7): 0 = Uncacheable (UC) 6 = Write-back (WB) Other values are reserved. ¹
5:3	This value is 1 less than the EPT page-walk length (see Section 31.3.2)
6	Setting this control to 1 enables accessed and dirty flags for EPT (see Section 31.3.5) ²
7	Setting this control to 1 enables enforcement of access rights for supervisor shadow-stack pages (see Section 31.3.3.2) ³
11:8	Reserved
M-1:12	Bits M-1:12 of the physical address of the 4-KByte aligned EPT paging-structure (an EPT PML4 table with 4-level EPT and an EPT PML5 table with 5-level EPT) ⁴
63:M	Reserved

NOTES:

1. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine what EPT paging-structure memory types are supported.
2. Not all processors support accessed and dirty flags for EPT. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.
3. Not all processors enforce access rights for shadow-stack pages. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.
4. M is an abbreviation for MAXPHYADDR. See footnote 2 on page 26-1.

The EPTP exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.

27.6.12 Virtual-Processor Identifier (VPID)

The **virtual-processor identifier** (VPID) is a 16-bit field. It exists only on processors that support the 1-setting of the “enable VPID” VM-execution control. See Section 31.1 for details regarding the use of this field.

27.6.13 Controls for PAUSE-Loop Exiting

On processors that support the 1-setting of the “PAUSE-loop exiting” VM-execution control, the VM-execution control fields include the following 32-bit fields:

- **PLE_Gap.** Software can configure this field as an upper bound on the amount of time between two successive executions of PAUSE in a loop.
- **PLE_Window.** Software can configure this field as an upper bound on the amount of time a guest is allowed to execute in a PAUSE loop.

These fields measure time based on a counter that runs at the same rate as the timestamp counter (TSC). See Section 28.1.3 for more details regarding PAUSE-loop exiting.

27.6.14 VM-Function Controls

The **VM-function controls** constitute a 64-bit vector that governs use of the VMFUNC instruction in VMX non-root operation. This field is supported only on processors that support the 1-settings of both the “activate secondary controls” primary processor-based VM-execution control and the “enable VM functions” secondary processor-based VM-execution control.

Table 27-10 lists the VM-function controls. See Section 28.5.6 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 27-10. Definitions of VM-Function Controls

Bit Position(s)	Name	Description
0	EPTP switching	The EPTP-switching VM function changes the EPT pointer to a value chosen from the EPTP list. See Section 28.5.6.3.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_VMFUNC (see Appendix A.11) to determine which bits are reserved. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 29.2.1.1).

Processors that support the 1-setting of the “EPTP switching” VM-function control also support a 64-bit field called the **EPTP-list address**. This field contains the physical address of the 4-KByte **EPTP list**. The EPTP list comprises 512 8-Byte entries (each an EPTP value) and is used by the EPTP-switching VM function (see Section 28.5.6.3).

27.6.15 VMCS Shadowing Bitmap Addresses

On processors that support the 1-setting of the “VMCS shadowing” VM-execution control, the VM-execution control fields include the 64-bit physical addresses of the **VMREAD bitmap** and the **VMWRITE bitmap**. Each bitmap is 4 KBytes in size and thus contains 32 KBits. The addresses are the **VMREAD-bitmap address** and the **VMWRITE-bitmap address**.

If the “VMCS shadowing” VM-execution control is 1, executions of VMREAD and VMWRITE may consult these bitmaps (see Section 27.10 and Section 33.3).

27.6.16 ENCLS-Exiting Bitmap

The **ENCLS-exiting bitmap** is a 64-bit field. If the “enable ENCLS exiting” VM-execution control is 1, execution of ENCLS causes a VM exit if the bit in this field corresponding to the value of EAX is 1. If the bit is 0, the instruction executes normally. See Section 28.1.3 for more information.

27.6.17 PCONFIG-Exiting Bitmap

The **PCONFIG-exiting bitmap** is a 64-bit field. If the “enable PCONFIG” VM-execution control is 1, execution of PCONFIG causes a VM exit if the bit in this field corresponding to the value of EAX is 1. If the control is 0, any execution of PCONFIG causes a #UD. See Section 28.1.3 for more information.

27.6.18 Control Field for Page-Modification Logging

The **PML address** is a 64-bit field. It is the 4-KByte aligned address of the **page-modification log**. The page-modification log consists of 512 64-bit entries. It is used for the page-modification logging feature. Details of the page-modification logging are given in Section 31.3.6.

If the “enable PML” VM-execution control is 1, VM entry ensures that the PML address is 4-KByte aligned. The PML address exists only on processors that support the 1-setting of the “enable PML” VM-execution control.

27.6.19 Controls for Virtualization Exceptions

On processors that support the 1-setting of the “EPT-violation #VE” VM-execution control, the VM-execution control fields include the following:

- **Virtualization-exception information address** (64 bits). This field contains the physical address of the **virtualization-exception information area**. When a logical processor encounters a virtualization exception, it saves virtualization-exception information at the virtualization-exception information address; see Section 28.5.7.2.
- **EPTP index** (16 bits). When an EPT violation causes a virtualization exception, the processor writes the value of this field to the virtualization-exception information area. The EPTP-switching VM function updates this field (see Section 28.5.6.3).

27.6.20 XSS-Exiting Bitmap

On processors that support the 1-setting of the “enable XSAVES/XRSTORS” VM-execution control, the VM-execution control fields include a 64-bit **XSS-exiting bitmap**. If the “enable XSAVES/XRSTORS” VM-execution control is 1, executions of XSAVES and XRSTORS may consult this bitmap (see Section 28.1.3 and Section 28.3).

27.6.21 Sub-Page-Permission-Table Pointer (SPPTP)

If the sub-page write-permission feature of EPT is enabled, EPT write permissions may be determined at a 128-byte granularity (see Section 31.3.4). These permissions are determined using a hierarchy of sub-page-permission structures in memory.

The root of this hierarchy is referenced by a VM-execution control field called the **sub-page-permission-table pointer** (SPPTP). The SPPTP contains the address of the base of the root SPP table (see Section 31.3.4.2). The format of this field is shown in Table 27-9.

Table 27-11. Format of Sub-Page-Permission-Table Pointer

Bit Position(s)	Field
11:0	Reserved
M-1:12	Bits N-1:12 of the physical address of the 4-KByte aligned root SPP table ¹
63:M	Reserved

NOTES:

1. M is an abbreviation for MAXPHYADDR. See footnote 2 on page 26-1.

The SPPTP exists only on processors that support the 1-setting of the “sub-page write permissions for EPT” VM-execution control.

27.6.22 Fields Related to Hypervisor-Managed Linear-Address Translation

Two fields are used when the “enable HLAT” VM-execution control is 1, enabling HLAT paging:

- The **hypervisor-managed linear-address translation pointer** (HLAT pointer or HLATP) is used by HLAT paging to locate and access the first paging structure used for linear-address translation (see Section 5.5). The format of this field is shown in Table 27-12.

Table 27-12. Format of Hypervisor-Managed Linear-Address Translation Pointer

Bit Position(s)	Field
2:0	Reserved
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the first HLAT paging structure during linear-address translation.
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the first HLAT paging structure during linear-address translation.
11:5	Reserved
M-1:12	Guest-physical address (4KB-aligned) of the first HLAT paging structure during linear-address translation. ¹
63:M	Reserved

NOTES:

1. M is an abbreviation for MAXPHYADDR. See footnote 2 on page 26-1.

- The HLAT prefix size. The value of this field determines which linear address are subject to HLAT paging. See Section 5.5.1.

These fields exist only on processors that support the 1-setting of the “enable HLAT” VM-execution control.

27.6.23 Fields Related to PASID Translation

Two 64-bit VM-execution control fields are used when the “PASID translation” VM-execution control is 1, enabling translation of PASIDs for executions of ENQCMD and ENQCMLS: the **low PASID directory address** and the **high PASID directory address**. These are the physical addresses of the low PASID directory and the high PASID directory, respectively. These fields exist only on processors that support the 1-setting of the “PASID translation” VM-execution control.

See Section 28.5.8 for information on the PASID-translation process for ENQCMD and ENQCMLS.

27.6.24 Instruction-Timeout Control

On processors that support the 1-setting of the “instruction timeout” VM-execution control, the VM-execution control fields include a 32-bit **instruction-timeout control**. The processor interprets the value of this field as an amount of time as measured in units of crystal clock cycles.¹ If the “instruction timeout” VM-execution control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within this amount of time.

27.6.25 Fields Controlling Virtualization of the IA32_SPEC_CTRL MSR

On processors that support the 1-setting of the “virtualize IA32_SPEC_CTRL” VM-execution control, the VM-execution control fields include the following 64-bit fields:

1. CPUID.15H:ECX enumerates the nominal frequency of the core crystal clock in Hz.

- **IA32_SPEC_CTRL mask.** Setting a bit in this field prevents guest software from modifying the corresponding bit in the IA32_SPEC_CTRL MSR.
- **IA32_SPEC_CTRL shadow.** This field contains the value that guest software expects to be in the IA32_SPEC_CTRL MSR.

Section 28.3 discusses how these fields are used in VMX non-root operation.

27.6.26 Fields Controlling SEAM Non-Root Operation

On processors that support the 1-setting of the “SEAM guest-physical address width” VM-execution control, the VM-execution control fields include the following fields:

- **SEAM-guest KeyID.** This 32-bit field contains the key identifier associated with the key used to encrypt SEAM private memory for the current SEAM guest.
- **SEAM shared EPT pointer.** This 64-bit field contains the address of the EPT paging structure that is the root of the hierarchy of structures that control the translation of shared guest-physical addresses. The format of this field is shown in Table 27-13.

Table 27-13. Format of SEAM Shared EPT Pointer

Bit Position(s)	Field
11:0	Reserved
M-1:12	Bits M-1:12 of the physical address of the 4-KByte aligned EPT paging-structure (an EPT PML4 table with 4-level EPT and an EPT PML5 table with 5-level EPT) ¹
63:M	Reserved

NOTES:

1. M is the value enumerated by CPUID.80000008H:EAX[7:0].

These fields control SEAM non-root operation. They are ignored outside of SEAM.

27.7 VM-EXIT CONTROL FIELDS

The VM-exit control fields govern the behavior of VM exits. They are discussed in Section 27.7.1 and Section 27.7.2.

27.7.1 VM-Exit Controls

The VM-exit controls constitute two vectors that govern the basic operation of VM exits. These are the **primary VM-exit controls** (32 bits) and the **secondary VM-exits controls** (64 bits).

Table 27-14 lists the primary VM-exit controls. See Chapter 29 for complete details of how these controls affect VM exits.

Table 27-14. Definitions of Primary VM-Exit Controls

Bit Position(s)	Name	Description
2	Save debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are saved on VM exit. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.

Table 27-14. Definitions of Primary VM-Exit Controls (Contd.)

Bit Position(s)	Name	Description
9	Host address-space size	On processors that support Intel 64 architecture, this control determines whether a logical processor is in 64-bit mode after the next VM exit. Its value is loaded into CS.L, IA32_EFER.LME, and IA32_EFER.LMA on every VM exit. ¹ This control must be 0 on processors that do not support Intel 64 architecture.
12	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM exit.
15	Acknowledge interrupt on exit	This control affects VM exits due to external interrupts: ▪ If such a VM exit occurs and this control is 1, the logical processor acknowledges the interrupt controller, acquiring the interrupt's vector. The vector is stored in the exiting-event identification field, which is marked valid. ▪ If such a VM exit occurs and this control is 0, the interrupt is not acknowledged and the exiting-event identification field is marked invalid.
18	Save IA32_PAT	This control determines whether the IA32_PAT MSR is saved on VM exit.
19	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM exit.
20	Save IA32_EFER	This control determines whether the IA32_EFER MSR is saved on VM exit.
21	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM exit.
22	Save VMX-preemption timer value	This control determines whether the value of the VMX-preemption timer is saved on VM exit.
23	Clear IA32_BNDCFGS	This control determines whether the IA32_BNDCFGS MSR is cleared on VM exit.
24	Conceal VMX from PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM exit or a VMCS packet on an SMM VM exit (see Chapter 36).
25	Clear IA32_RTIT_CTL	This control determines whether the IA32_RTIT_CTL MSR is cleared on VM exit.
26	Clear IA32_LBR_CTL	This control determines whether the IA32_LBR_CTL MSR is cleared on VM exit.
27	Clear UINV	This control determines whether UINV is cleared on VM exit.
28	Load CET state	This control determines whether CET-related MSRs and SSP are loaded on VM exit.
29	Load PKRS	This control determines whether the IA32_PKRS MSR is loaded on VM exit.
30	Save IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is saved on VM exit.
31	Activate secondary controls	This control determines whether the secondary VM-exit controls are used. If this control is 0, the logical processor operates as if all the secondary VM-exit controls were also 0.

NOTES:

1. Since the Intel 64 architecture specifies that IA32_EFER.LMA is always set to the logical-AND of CR0.PG and IA32_EFER.LME, and since CR0.PG is always 1 in VMX root operation, IA32_EFER.LMA is always identical to IA32_EFER.LME in VMX root operation.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_EXIT_CTLS and IA32_VMX_TRUE_EXIT_CTLS (see Appendix A.4) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 29.2.1.2).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8, 10, 11, 13, 14, 16, and 17. The VMX capability MSR IA32_VMX_EXIT_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_EXIT_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-exit controls determines whether the secondary VM-exit controls are used. If that bit is 0, VM entries and VM exits function as if all the secondary VM-exit controls were 0. Processors

that support only the 0-setting of bit 31 of the primary VM-exit controls do not support the secondary VM-exit controls.

Table 27-15 lists the secondary VM-exit controls. See Chapter 29 for more details of how these controls affect VM exits.

Table 27-15. Definitions of Secondary VM-Exit Controls

Bit Position(s)	Name	Description
0	Save FRED	This control determines whether the FRED MSRs are saved on VM exit.
1	Load FRED	This control determines whether the FRED MSRs are loaded on VM exit.
3	Prematurely busy shadow stack	If this control is 1, VM exits that cause a shadow stack to become prematurely busy (see Section 18.2.3, “Supervisor Shadow Stack Token,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1) indicate this fact and save additional information into the VMCS.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_EXIT_CTL2 (see Appendix A.4.2) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 29.2.1.2).

27.7.2 VM-Exit Controls for MSRs

A VMM may specify lists of MSRs to be stored and loaded on VM exits. The following VM-exit control fields determine how MSRs are stored on VM exits:

- **VM-exit MSR-store count** (32 bits). This field specifies the number of MSRs to be stored on VM exit. It is recommended that this count not exceed 512.¹ Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.
- **VM-exit MSR-store address** (64 bits). This field contains the physical address of the VM-exit MSR-store area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-store count. The format of each entry is given in Table 27-16. If the VM-exit MSR-store count is not zero, the address must be 16-byte aligned.

Table 27-16. Format of an MSR Entry

Bit Position(s)	Contents
31:0	MSR index
63:32	Reserved
127:64	MSR data

See Section 30.4 for how this area is used on VM exits.

The following VM-exit control fields determine how MSRs are loaded on VM exits:

- **VM-exit MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM exit. It is recommended that this count not exceed 512. Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.²
- **VM-exit MSR-load address** (64 bits). This field contains the physical address of the VM-exit MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-load count (see Table 27-16). If the VM-exit MSR-load count is not zero, the address must be 16-byte aligned.

See Section 30.6 for how this area is used on VM exits.

1. Future implementations may allow more MSRs to be stored reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).
2. Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

27.8 VM-ENTRY CONTROL FIELDS

The VM-entry control fields govern the behavior of VM entries. They are discussed in Sections 27.8.1 through 27.8.3.

27.8.1 VM-Entry Controls

The **VM-entry controls** constitute a 32-bit vector that governs the basic operation of VM entries. Table 27-17 lists the controls supported. See Chapter 27 for how these controls affect VM entries.

Table 27-17. Definitions of VM-Entry Controls

Bit Position(s)	Name	Description
2	Load debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are loaded on VM entry. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	IA-32e mode guest	On processors that support Intel 64 architecture, this control determines whether the logical processor is in IA-32e mode after VM entry. Its value is loaded into IA32_EFER.LMA as part of VM entry. ¹ This control must be 0 on processors that do not support Intel 64 architecture.
10	Entry to SMM	This control determines whether the logical processor is in system-management mode (SMM) after VM entry. This control must be 0 for any VM entry from outside SMM.
11	Deactivate dual-monitor treatment	If set to 1, the default treatment of SMIs and SMM is in effect after the VM entry (see Section 34.15.7). This control must be 0 for any VM entry from outside SMM.
13	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM entry.
14	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM entry.
15	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM entry.
16	Load IA32_BNDCFGS	This control determines whether the IA32_BNDCFGS MSR is loaded on VM entry.
17	Conceal VMX from PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM entry or a VMCS packet on a VM entry that returns from SMM (see Chapter 36).
18	Load IA32_RTIT_CTL	This control determines whether the IA32_RTIT_CTL MSR is loaded on VM entry.
19	Load UINV	This control determines whether UINV is loaded on VM entry.
20	Load CET state	This control determines whether CET-related MSRs and SSP are loaded on VM entry.
21	Load guest IA32_LBR_CTL	This control determines whether the IA32_LBR_CTL MSR is loaded on VM entry.
22	Load PKRS	This control determines whether the IA32_PKRS MSR is loaded on VM entry.
23	Load FRED	This control determines whether the FRED MSRs are loaded on VM entry.
25	Allow SEAM-guest telemetry	If set to 1, core out-of-band telemetry can be enabled upon VM entry to SEAM non-root operation. See Section 20.21.

NOTES:

1. Bit 5 of the IA32_VMX_MISC MSR is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control. If it is read as 1, every VM exit stores the value of IA32_EFER.LMA into the “IA-32e mode guest” VM-entry control (see Section 30.2).

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_ENTRY_CTLS and IA32_VMX_TRUE_ENTRY_CTLS (see Appendix A.5) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 29.2.1.3).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8 and 12. The VMX capability MSR IA32_VMX_ENTRY_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_ENTRY_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

27.8.2 VM-Entry Controls for MSRs

A VMM may specify a list of MSRs to be loaded on VM entries. The following VM-entry control fields manage this functionality:

- **VM-entry MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM entry. It is recommended that this count not exceed 512. Otherwise, unpredictable processor behavior (including a machine check) may result during VM entry.¹
- **VM-entry MSR-load address** (64 bits). This field contains the physical address of the VM-entry MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-entry MSR-load count. The format of entries is described in Table 27-16. If the VM-entry MSR-load count is not zero, the address must be 16-byte aligned.

See Section 29.4 for details of how this area is used on VM entries.

27.8.3 VM-Entry Controls for Event Injection

VM entry can be configured to conclude by delivering an event (after all guest state and MSRs have been loaded). This process is called **event injection** and is controlled by the following three VM-entry control fields:

- **Injected-event identification field** (32 bits). This field provides details about the event to be injected. Table 27-18 describes the field.²

Table 27-18. Format of the Injected-Event Identification Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Event type: ¹ 0: External interrupt 1: Reserved 2: Non-maskable interrupt (NMI) 3: Hardware exception (e.g., #PF) 4: Software interrupt (INT <i>n</i>) 5: Privileged software exception (INT1) 6: Software exception (INT3 or INTO) 7: Other event
11	Deliver error code (0 = do not deliver; 1 = deliver)
12	Reserved
13	Nested exception (may be set only if IA32_VMX_BASIC[58] = 1)
30:14	Reserved
31	Valid

NOTES:

1. Older versions of this document called this sub-field interruption type.

— The **vector** (bits 7:0) identifies the interrupt or exception or which other event is injected.

1. Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

2. Older versions of this document called this field VM-entry interruption information.

- The **event type** (bits 10:8) determines details of how the injection is performed. In general, a VMM should use the type hardware exception for all exceptions **other than** the following:
 - breakpoint exceptions (#BP; a VMM should use the type software exception);
 - overflow exceptions (#OF a VMM should use the use type software exception); and
 - those debug exceptions (#DB) that are generated by INT1 (a VMM should use the use type privileged software exception).¹

The type **other event** is used for injection of events that are not delivered through the IDT.²

- For exceptions, the **deliver-error-code bit** (bit 11) determines whether delivery pushes an error code on the guest stack.
- For exceptions, the **nested-exception bit** (bit 13) indicates that VM entry should treat the injected event as an exception nested on delivery of another event. This bit may be set only if `IA32_VMX_BASIC[58] = 1`.
- VM entry injects an event if and only if the **valid bit** (bit 31) is 1. The valid bit in this field is cleared on every VM exit (see Section 30.2).

- **Injected-event exception error code** (32 bits). This field is used if and only if the valid bit (bit 31) and the deliver-error-code bit (bit 11) are both set in the **injected-event identification** field.³
- **Injected-event data** (64 bits). This field is used if and only if the valid bit is set in the injected-event identification field and FRED transitions will be enabled following VM entry.
- **VM-entry instruction length** (32 bits). For injection of events whose type is software interrupt, software exception, or privileged software exception, this field is used to determine the value of RIP that is pushed on the stack. It is used also for injection of SYSCALL and SYSEXIT when FRED transitions would be enabled.

See Section 29.6 for details regarding the mechanics of event injection, including the use of the **event type**, the **nested-exception bit**, and the VM-entry instruction length.

VM exits clear the valid bit (bit 31) in the **injected-event identification** field.

27.9 VM-EXIT INFORMATION FIELDS

The VMCS contains a section of fields that contain information about the most recent VM exit.

On some processors, attempts to write to these fields with VMWRITE fail (see “VMWRITE—Write Field to Virtual-Machine Control Structure” in Chapter 32).⁴

27.9.1 Basic VM-Exit Information

The following VM-exit information fields provide basic information about a VM exit:

- **Exit reason** (32 bits). This field encodes the reason for the VM exit and has the structure given in Table 27-19.

Table 27-19. Format of Exit Reason

Bit Position(s)	Contents
15:0	Basic exit reason.
16	Always cleared to 0.
24:17	Not currently defined.

1. The type hardware exception should be used for all other debug exceptions.

2. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with values 1 or 3 for *n*.

3. Older versions of this document called this field VM-entry exception error code.

4. Software can discover whether these fields can be written by reading the VMX capability MSR `IA32_VMX_MISC` (see Appendix A.6).

Table 27-19. Format of Exit Reason (Contd.)

Bit Position(s)	Contents
25	A VM exit saves this bit as 1 to indicate that the VM exit caused a shadow stack to become prematurely busy.
26	A VM exit saves this bit as 1 to indicate that the VM exit occurred after assertion of a bus lock while the “VMM bus-lock detection” VM-execution control was 1.
27	A VM exit saves this bit as 1 to indicate that the VM exit was incident to enclave mode.
28	Pending MTF VM exit.
29	VM exit from VMX root operation.
30	Not currently defined.
31	VM-entry failure (0 = true VM exit; 1 = VM-entry failure)

- Bits 15:0 provide basic information about the cause of the VM exit (if bit 31 is clear) or of the VM-entry failure (if bit 31 is set). Appendix C enumerates the basic exit reasons.
- Bit 16 is always cleared to 0.
- Bit 25 is set to 1 if the “prematurely busy shadow stack” VM-exit control is 1 and the VM exit caused a shadow stack to become prematurely busy (see Section 28.4.3). Otherwise, the bit is cleared.
- Bit 26 is set to 1 if the VM exit occurred after assertion of a bus lock while the “VMM bus-lock detection” VM-execution control was 1. Such VM exits include those that occur due to the 1-setting of that control as well as others that might occur during execution of an instruction that asserted a bus lock.
- Bit 27 is set to 1 if the VM exit occurred while the logical processor was in enclave mode. See Section 30.2.1 for details.
- Bit 28 is set only by an SMM VM exit (see Section 34.15.2) that took priority over an MTF VM exit (see Section 28.5.2) that would have occurred had the SMM VM exit not occurred. See Section 34.15.2.3.
- Bit 29 is set if and only if the processor was in VMX root operation at the time the VM exit occurred. This can happen only for SMM VM exits. See Section 34.15.2.
- Because some VM-entry failures load processor state from the host-state area (see Section 29.8), software must be able to distinguish such cases from true VM exits. Bit 31 is used for that purpose.
- **Exit qualification** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field contains additional information about the cause of VM exits due to the following: debug exceptions; page-fault exceptions; start-up IPIs (SIPs); task switches; INVEPT; INVLPID; INVVPID; LGDT; LIDT; LLDT; LTR; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; XRSTORS; XSAVES; control-register accesses; MOV DR; I/O instructions; and MWAIT. The format of the field depends on the cause of the VM exit. See Section 30.2.1 for details.
- **Guest-linear address** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is used in the following cases:
 - VM exits due to attempts to execute LMSW with a memory operand.
 - VM exits due to attempts to execute INS or OUTS.
 - VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions.
 - Certain VM exits due to EPT violations
 See Section 30.2.1 and Section 34.15.2.3 for details of when and how this field is used.
- **Guest-physical address** (64 bits). This field is used by VM exits due to EPT violations and EPT misconfigurations. See Section 30.2.1 for details of when and how this field is used.

27.9.2 Information for VM Exits Due to Vectored Events

Event-specific information is provided for VM exits due to the following vectored events: exceptions (including those generated by the instructions INT3, INTO, INT1, BOUND, UD0, UD1, UD2, and UDB); external interrupts that occur while the “acknowledge interrupt on exit” VM-exit control is 1; and non-maskable interrupts (NMIs). This information is provided in the following fields:

- **Exiting-event identification** (32 bits). This field receives basic information associated with the event causing the VM exit. Table 27-20 describes this field.¹

Table 27-20. Format of the Exiting-Event Identification Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Event type: ¹ 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Not used 5: Privileged software exception 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
12	NMI unblocking due to IRET
13	Nested exception (during FRED event delivery)
30:14	Not currently defined
31	Valid

NOTES:

1. Older versions of this document called this sub-field interruption type.

- **Exiting-event error code** (32 bits). For VM exits caused by hardware exceptions that would have delivered an error code on the stack, this field receives that error code.²

Section 30.2.2 provides details of how these fields are saved on VM exits.

27.9.3 Information for VM Exits That Occur During Event Delivery

Additional information is provided for VM exits that occur during event delivery in VMX non-root operation.³ This information is provided in the following fields:

- **Original-event identification** (32 bits). This field receives basic information associated with the event that was being delivered when the VM exit occurred. Table 27-21 describes this field.⁴

Table 27-21. Format of the Original-Event Identification Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception

1. Older versions of this document called this field VM-exit interruption information.

2. Older versions of this document called this field VM-exit interruption error code.

3. This includes cases in which the event delivery was caused by event injection as part of VM entry; see Section 29.6.1.2.

4. Older versions of this document called this field IDT-vectoring information.

Table 27-21. Format of the Original-Event Identification Field (Contd.)

Bit Position(s)	Content
10:8	Event type: ¹ 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Software interrupt 5: Privileged software exception 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
12	Not currently defined
13	Exception nested on FRED event delivery
30:14	Not currently defined
31	Valid

NOTES:

1. Older versions of this document called this sub-field interruption type.

- **Original-event error code** (32 bits). For VM exits that occur during delivery of hardware exceptions that would have delivered an error code on the stack, this field receives that error code.¹
- **Original-event data** (64 bits). For VM exits that occur during FRED event delivery, this field receives the event data that that delivery would have saved on the stack.

See Section 30.2.4 provides details of how these fields are saved on VM exits.

27.9.4 Information for VM Exits Due to Instruction Execution

The following fields are used for VM exits caused by attempts to execute certain instructions in VMX non-root operation:

- **VM-exit instruction length** (32 bits). For VM exits resulting from instruction execution, this field receives the length in bytes of the instruction whose execution led to the VM exit.² See Section 30.2.5 for details of when and how this field is used.
- **VM-exit instruction information** (32 bits). This field is used for VM exits due to attempts to execute INS, INVEPT, INVVPID, LIDT, LGDT, LLDT, LTR, OUTS, SIDT, SGDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, or VMXON.³ The format of the field depends on the cause of the VM exit. See Section 30.2.5 for details.

The following fields (64 bits each; 32 bits on processors that do not support Intel 64 architecture) are used only for VM exits due to SMIs that arrive immediately after retirement of I/O instructions. They provide information about that I/O instruction:

- **I/O RCX**. The value of RCX before the I/O instruction started.
- **I/O RSI**. The value of RSI before the I/O instruction started.
- **I/O RDI**. The value of RDI before the I/O instruction started.
- **I/O RIP**. The value of RIP before the I/O instruction started (the RIP that addressed the I/O instruction).

1. Older versions of this document called this field IDT-vectoring error code.

2. This field is also used for VM exits that occur during the delivery of a software interrupt, software exception, and (if FRED transitions are enabled), SYSCALL and SYSENTER.

3. Whether the processor provides this information on VM exits due to attempts to execute INS or OUTS can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

An execution of WRMSRLIST causes a VM exit if it writes to an MSR that cannot be written due to the contents of the MSR bitmaps (see Section 28.1.3). Such VM exits save the data that would have been written to the MSR in a 64-bit field called **MSR data**. This field is supported only on processors that support the 1-setting of the “enable MSR-list instructions” VM-execution control.

27.9.5 VM-Instruction Error Field

The 32-bit **VM-instruction error field** does not provide information about the most recent VM exit. In fact, it is not modified on VM exits. Instead, it provides information about errors encountered by a non-faulting execution of one of the VMX instructions.

27.10 VMCS TYPES: ORDINARY AND SHADOW

Every VMCS is either an **ordinary VMCS** or a **shadow VMCS**. A VMCS’s type is determined by the shadow-VMCS indicator in the VMCS region (this is the value of bit 31 of the first 4 bytes of the VMCS region; see Table 27-1): 0 indicates an ordinary VMCS, while 1 indicates a shadow VMCS. Shadow VMCSs are supported only on processors that support the 1-setting of the “VMCS shadowing” VM-execution control (see Section 27.6.2).

A shadow VMCS differs from an ordinary VMCS in two ways:

- An ordinary VMCS can be used for VM entry but a shadow VMCS cannot. Attempts to perform VM entry when the current VMCS is a shadow VMCS fail (see Section 29.1).
- The VMREAD and VMWRITE instructions can be used in VMX non-root operation to access a shadow VMCS but not an ordinary VMCS. This fact results from the following:
 - If the “VMCS shadowing” VM-execution control is 0, execution of the VMREAD and VMWRITE instructions in VMX non-root operation always cause VM exits (see Section 28.1.3).
 - If the “VMCS shadowing” VM-execution control is 1, execution of the VMREAD and VMWRITE instructions in VMX non-root operation can access the VMCS referenced by the VMCS link pointer (see Section 33.3).
 - If the “VMCS shadowing” VM-execution control is 1, VM entry ensures that any VMCS referenced by the VMCS link pointer is a shadow VMCS (see Section 29.3.1.5).

In VMX root operation, both types of VMCSs can be accessed with the VMREAD and VMWRITE instructions.

Software should not modify the shadow-VMCS indicator in the VMCS region of a VMCS that is active. Doing so may cause the VMCS to become corrupted (see Section 27.11.1). Before modifying the shadow-VMCS indicator, software should execute VMCLEAR for the VMCS to ensure that it is not active.

27.11 SOFTWARE USE OF THE VMCS AND RELATED STRUCTURES

This section details guidelines that software should observe when using a VMCS and related structures. It also provides descriptions of consequences for failing to follow guidelines.

27.11.1 Software Use of Virtual-Machine Control Structures

To ensure proper processor behavior, software should observe certain guidelines when using an active VMCS.

No VMCS should ever be active on more than one logical processor. If a VMCS is to be “migrated” from one logical processor to another, the first logical processor should execute VMCLEAR for the VMCS (to make it inactive on that logical processor and to ensure that all VMCS data are in memory) before the other logical processor executes VMPTRLD for the VMCS (to make it active on the second logical processor).¹ A VMCS that is made active on more than one logical processor may become **corrupted** (see below).

1. As noted in Section 27.1, execution of the VMPTRLD instruction makes a VMCS active. In addition, VM entry makes active any shadow VMCS referenced by the VMCS link pointer in the current VMCS. If a shadow VMCS is made active by VM entry, it is necessary to execute VMCLEAR for that VMCS before allowing that VMCS to become active on another logical processor.

Software should not modify the shadow-VMCS indicator (see Table 27-1) in the VMCS region of a VMCS that is active. Doing so may cause the VMCS to become corrupted. Before modifying the shadow-VMCS indicator, software should execute VMCLEAR for the VMCS to ensure that it is not active.

Software should use the VMREAD and VMWRITE instructions to access the different fields in the current VMCS (see Section 27.11.2). Software should never access or modify the VMCS data of an active VMCS using ordinary memory operations, in part because the format used to store the VMCS data is implementation-specific and not architecturally defined, and also because a logical processor may maintain some VMCS data of an active VMCS on the processor and not in the VMCS region. The following items detail some of the hazards of accessing VMCS data using ordinary memory operations:

- Any data read from a VMCS with an ordinary memory read does not reliably reflect the state of the VMCS. Results may vary from time to time or from logical processor to logical processor.
- Writing to a VMCS with an ordinary memory write is not guaranteed to have a deterministic effect on the VMCS. Doing so may cause the VMCS to become corrupted (see below).

(Software can avoid these hazards by removing any linear-address mappings to a VMCS region before executing a VMPTRLD for that region and by not remapping it until after executing VMCLEAR for that region.)

If a logical processor leaves VMX operation, any VMCSs active on that logical processor may be corrupted (see below). To prevent such corruption of a VMCS that may be used either after a return to VMX operation or on another logical processor, software should execute VMCLEAR for that VMCS before executing the VMXOFF instruction or removing power from the processor (e.g., as part of a transition to the S3 and S4 power states).

This section has identified operations that may cause a VMCS to become corrupted. These operations may cause the VMCS's data to become undefined. Behavior may be unpredictable if that VMCS used subsequently on any logical processor. The following items detail some hazards of VMCS corruption:

- VM entries may fail for unexplained reasons or may load undesired processor state.
- The processor may not correctly support VMX non-root operation as documented in Chapter 27 and may generate unexpected VM exits.
- VM exits may load undesired processor state, save incorrect state into the VMCS, or cause the logical processor to transition to a shutdown state.

27.11.2 VMREAD, VMWRITE, and Encodings of VMCS Fields

Every field of the VMCS is associated with a 32-bit value that is its **encoding**. The encoding is provided in an operand to VMREAD and VMWRITE when software wishes to read or write that field. These instructions fail if given, in 64-bit mode, an operand that sets an encoding bit beyond bit 32. See Chapter 32 for a description of these instructions.

The structure of the 32-bit encodings of the VMCS components is determined principally by the width of the fields and their function in the VMCS. See Table 27-22.

Table 27-22. Structure of VMCS Component Encoding

Bit Position(s)	Contents
0	Access type (0 = full; 1 = high); must be full for 16-bit, 32-bit, and natural-width fields
9:1	Index
11:10	Type: 0: control 1: VM-exit information 2: guest state 3: host state
12	Reserved (must be 0)

Table 27-22. Structure of VMCS Component Encoding (Contd.)

Bit Position(s)	Contents
14:13	Width: 0: 16-bit 1: 64-bit 2: 32-bit 3: natural-width
31:15	Reserved (must be 0)

The following items detail the meaning of the bits in each encoding:

- **Field width.** Bits 14:13 encode the width of the field.
 - A value of 0 indicates a 16-bit field.
 - A value of 1 indicates a 64-bit field.
 - A value of 2 indicates a 32-bit field.
 - A value of 3 indicates a **natural-width** field. Such fields have 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

Fields whose encodings use value 1 are specially treated to allow 32-bit software access to all 64 bits of the field. Such access is allowed by defining, for each such field, an encoding that allows direct access to the high 32 bits of the field. See below.
- **Field type.** Bits 11:10 encode the type of VMCS field: control, guest-state, host-state, or VM-exit information. (The last category also includes the VM-instruction error field.)
- **Index.** Bits 9:1 distinguish components with the same field width and type.
- **Access type.** Bit 0 must be 0 for all fields except for 64-bit fields (those with field-width 1; see above). A VMREAD or VMWRITE using an encoding with this bit cleared to 0 accesses the entire field. For a 64-bit field with field-width 1, a VMREAD or VMWRITE using an encoding with this bit set to 1 accesses only the high 32 bits of the field.

Appendix B gives the encodings of all fields in the VMCS.

The following describes the operation of VMREAD and VMWRITE based on processor mode, VMCS-field width, and access type:

- 16-bit fields:
 - A VMREAD returns the value of the field in bits 15:0 of the destination operand; other bits of the destination operand are cleared to 0.
 - A VMWRITE writes the value of bits 15:0 of the source operand into the VMCS field; other bits of the source operand are not used.
- 32-bit fields:
 - A VMREAD returns the value of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
 - A VMWRITE writes the value of bits 31:0 of the source operand into the VMCS field; in 64-bit mode, bits 63:32 of the source operand are not used.
- 64-bit fields and natural-width fields using the full access type outside IA-32e mode.
 - A VMREAD returns the value of bits 31:0 of the field in its destination operand; bits 63:32 of the field are ignored.
 - A VMWRITE writes the value of its source operand to bits 31:0 of the field and clears bits 63:32 of the field.
- 64-bit fields and natural-width fields using the full access type in 64-bit mode (only on processors that support Intel 64 architecture).
 - A VMREAD returns the value of the field in bits 63:0 of the destination operand

- A VMWRITE writes the value of bits 63:0 of the source operand into the VMCS field.
- 64-bit fields using the high access type.
 - A VMREAD returns the value of bits 63:32 of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
 - A VMWRITE writes the value of bits 31:0 of the source operand to bits 63:32 of the field; in 64-bit mode, bits 63:32 of the source operand are not used.

Software seeking to read a 64-bit field outside IA-32e mode can use VMREAD with the full access type (reading bits 31:0 of the field) and VMREAD with the high access type (reading bits 63:32 of the field); the order of the two VMREAD executions is not important. Software seeking to modify a 64-bit field outside IA-32e mode should first use VMWRITE with the full access type (establishing bits 31:0 of the field while clearing bits 63:32) and then use VMWRITE with the high access type (establishing bits 63:32 of the field).

27.11.3 Initializing a VMCS

Software should initialize fields in a VMCS (using VMWRITE) before using the VMCS for VM entry. Failure to do so may result in unpredictable behavior; for example, a VM entry may fail for unexplained reasons, or a successful transition (VM entry or VM exit) may load processor state with unexpected values.

It is not necessary to initialize fields that the logical processor will not use. (For example, it is not necessary to initialize the MSR-bitmap address if the “use MSR bitmaps” VM-execution control is 0.)

A processor maintains some VMCS information that cannot be modified with the VMWRITE instruction; this includes a VMCS’s launch state (see Section 27.1). Such information may be stored in the VMCS data portion of a VMCS region. Because the format of this information is implementation-specific, there is no way for software to know, when it first allocates a region of memory for use as a VMCS region, how the processor will determine this information from the contents of the memory region.

In addition to its other functions, the VMCLEAR instruction initializes any implementation-specific information in the VMCS region referenced by its operand. To avoid the uncertainties of implementation-specific behavior, software should execute VMCLEAR on a VMCS region before making the corresponding VMCS active with VMPTRLD for the first time. (Figure 27-1 illustrates how execution of VMCLEAR puts a VMCS into a well-defined state.)

The following software usage is consistent with these limitations:

- VMCLEAR should be executed for a VMCS before it is used for VM entry for the first time.
- VMLAUNCH should be used for the first VM entry using a VMCS after VMCLEAR has been executed for that VMCS.
- VMRESUME should be used for any subsequent VM entry using a VMCS (until the next execution of VMCLEAR for the VMCS).

It is expected that, in general, VMRESUME will have lower latency than VMLAUNCH. Since “migrating” a VMCS from one logical processor to another requires use of VMCLEAR (see Section 27.11.1), which sets the launch state of the VMCS to “clear”, such migration requires the next VM entry to be performed using VMLAUNCH. Software developers can avoid the performance cost of increased VM-entry latency by avoiding unnecessary migration of a VMCS from one logical processor to another.

27.11.4 Software Access to Related Structures

In addition to data in the VMCS region itself, VMX non-root operation can be controlled by data structures that are referenced by pointers in a VMCS (for example, the I/O bitmaps). While the pointers to these data structures are parts of the VMCS, the data structures themselves are not. They are not accessible using VMREAD and VMWRITE but by ordinary memory writes.

Software should ensure that each such data structure is modified only when no logical processor with a current VMCS that references it is in VMX non-root operation. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 27.11.1). Exceptions are made for the following data structures (subject to detailed discussion in the sections indicated): EPT paging structures and the data structures used to locate SPP

vectors (Section 31.4.3); the virtual-APIC page (Section 32.1); the posted interrupt descriptor (Section 32.6); and the virtualization-exception information area (Section 28.5.7.2).

27.11.5 VMXON Region

Before executing VMXON, software allocates a region of memory (called the VMXON region)¹ that the logical processor uses to support VMX operation. The physical address of this region (the VMXON pointer) is provided in an operand to VMXON. The VMXON pointer is subject to the limitations that apply to VMCS pointers:

- The VMXON pointer must be 4-KByte aligned (bits 11:0 must be zero).
- The VMXON pointer must not set any bits beyond the processor's physical-address width.^{2,3}

Before executing VMXON, software should write the VMCS revision identifier (see Section 27.2) to the VMXON region. (Specifically, it should write the 31-bit VMCS revision identifier to bits 30:0 of the first 4 bytes of the VMXON region; bit 31 should be cleared to 0.) It need not initialize the VMXON region in any other way. Software should use a separate region for each logical processor and should not access or modify the VMXON region of a logical processor between execution of VMXON and VMXOFF on that logical processor. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 27.11.1).

-
1. The amount of memory required for the VMXON region is the same as that required for a VMCS region. This size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).
 2. CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. It is used to determine the value of and MAXPHYADDR, which constrains the VMXON pointer. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36]. (This is different in secure arbitration mode — SEAM — but VMXON cannot be used in SEAM.) IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode SEAM-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for KeyIDs generally.
 3. If IA32_VMX_BASIC[48] is read as 1, the VMXON pointer must not set any bits in the range 63:32; see Appendix A.1.

22. Updates to Chapter 28, Volume 3C

Change bars and text show changes to Chapter 28 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- 23.** This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- 24.** Updated Section 28.2, "Other Causes of VM Exits" for FRED.
- 25.** In Section 28.3, "Changes to Instruction Behavior in VMX Non-Root Operation", added ERETS and ERETU for FRED and PBNDKB for Total Storage Encryption to list of instructions.
- 26.** Updated Section 28.4.2, "Treatment of Task Switches" for FRED replacing IDT-vectoring with original-event and VM-exit with exiting-event.
- 27.** Removed reference to IDT in Section 28.5.7.1, "Convertible EPT Violations".
- 28.** Updated Section 28.5.7.3, "Delivery of Virtualization Exceptions" for FRED replacing IDT-vectoring with original-event and VM-exit with exiting-event.
- 29.** Added PBNDKB to the list of instructions affected by VMX non-root operation in Section 28.3, "Changes to Instruction Behavior in VMX Non-Root Operation".

In a virtualized environment using VMX, the guest software stack typically runs on a logical processor in VMX non-root operation. This mode of operation is similar to that of ordinary processor operation outside of the virtualized environment. This chapter describes the differences between VMX non-root operation and ordinary processor operation with special attention to causes of VM exits (which bring a logical processor from VMX non-root operation to root operation). The differences between VMX non-root operation and ordinary processor operation are described in the following sections:

- Section 28.1, “Instructions That Cause VM Exits.”
- Section 28.2, “Other Causes of VM Exits.”
- Section 28.3, “Changes to Instruction Behavior in VMX Non-Root Operation.”
- Section 28.4, “Other Changes in VMX Non-Root Operation.”
- Section 28.5, “Features Specific to VMX Non-Root Operation.”
- Section 28.6, “Unrestricted Guests.”

Chapter 28, “VMX Non-Root Operation,” describes the data control structures that govern VMX non-root operation. Chapter 28, “VMX Non-Root Operation,” describes the operation of VM entries by which the processor transitions from VMX root operation to VMX non-root operation. Chapter 28, “VMX Non-Root Operation,” describes the operation of VM exits by which the processor transitions from VMX non-root operation to VMX root operation.

Chapter 31, “VMX Support for Address Translation,” describes two features that support address translation in VMX non-root operation. Chapter 32, “APIC Virtualization and Virtual Interrupts,” describes features that support virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC) in VMX non-root operation.

28.1 INSTRUCTIONS THAT CAUSE VM EXITS

Certain instructions may cause VM exits if executed in VMX non-root operation. Unless otherwise specified, such VM exits are “fault-like,” meaning that the instruction causing the VM exit does not execute and no processor state is updated by the instruction. Section 30.1 details architectural state in the context of a VM exit.

Section 28.1.1 defines the prioritization between faults and VM exits for instructions subject to both. Section 28.1.2 identifies instructions that cause VM exits whenever they are executed in VMX non-root operation (and thus can never be executed in VMX non-root operation). Section 28.1.3 identifies instructions that cause VM exits depending on the settings of certain VM-execution control fields (see Section 27.6).

28.1.1 Relative Priority of Faults and VM Exits

The following principles describe the ordering between existing faults and VM exits:

- Certain exceptions have priority over VM exits. These include invalid-opcode exceptions, faults based on privilege level,¹ and general-protection exceptions that are based on checking I/O permission bits in the task-state segment (TSS). For example, execution of RDMSR with CPL = 3 generates a general-protection exception and not a VM exit.²
- Faults incurred while fetching instruction operands have priority over VM exits that are conditioned based on the contents of those operands (see LMSW in Section 28.1.3).
- VM exits caused by execution of the INS and OUTS instructions (resulting either because the “unconditional I/O exiting” VM-execution control is 1 or because the “use I/O bitmaps control is 1”) have priority over the following faults:

1. These include faults generated by attempts to execute, in virtual-8086 mode, privileged instructions that are not recognized in that mode.
2. MOV DR is an exception to this rule; see Section 28.1.3.

- A general-protection fault due to the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) being unusable
- A general-protection fault due to an offset beyond the limit of the relevant segment
- An alignment-check exception
- Fault-like VM exits have priority over exceptions other than those mentioned above. For example, RDMSR of a non-existent MSR with CPL = 0 generates a VM exit and not a general-protection exception.

When Section 28.1.2 or Section 28.1.3 (below) identify an instruction execution that may lead to a VM exit, it is assumed that the instruction does not incur a fault that takes priority over a VM exit.

28.1.2 Instructions That Cause VM Exits Unconditionally

The following instructions cause VM exits when they are executed in VMX non-root operation: CPUID, GETSEC,¹ INVD, and XSETBV. This is also true of instructions introduced with VMX: INVEPT, INVVPID, SEAMCALL, TDCALL, VMCALL,² VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMRESUME, VMXOFF, and VMXON.

28.1.3 Instructions That Cause VM Exits Conditionally

Certain instructions cause VM exits in VMX non-root operation depending on the setting of the VM-execution controls. The following instructions can cause “fault-like” VM exits based on the conditions described:³

- **CLTS.** The CLTS instruction causes a VM exit if the bits in position 3 (corresponding to CR0.TS) are set in both the CR0 guest/host mask and the CR0 read shadow.
- **ENCLS.** The ENCLS instruction causes a VM exit if the “enable ENCLS exiting” VM-execution control is 1 and one of the following is true:
 - The value of EAX is less than 63 and the corresponding bit in the ENCLS-exiting bitmap is 1 (see Section 27.6.16).
 - The value of EAX is greater than or equal to 63 and bit 63 in the ENCLS-exiting bitmap is 1.
- **ENQCMD, ENQCMLS.** The behavior of each of these instructions is determined by the setting of the “PASID translation” VM-execution control. If that control is 0, the instruction executes normally. If the control is 1, instruction behavior is modified and may cause a VM exit. See Section 28.5.8.
- **HLT.** The HLT instruction causes a VM exit if the “HLT exiting” VM-execution control is 1.
- **IN, INS/INSB/INSD, OUT, OUTS/OUTSB/OUTSD.** The behavior of each of these instructions is determined by the settings of the “unconditional I/O exiting” and “use I/O bitmaps” VM-execution controls:
 - If both controls are 0, the instruction executes normally.
 - If the “unconditional I/O exiting” VM-execution control is 1 and the “use I/O bitmaps” VM-execution control is 0, the instruction causes a VM exit.
 - If the “use I/O bitmaps” VM-execution control is 1, the instruction causes a VM exit if it attempts to access an I/O port corresponding to a bit set to 1 in the appropriate I/O bitmap (see Section 27.6.4). If an I/O operation “wraps around” the 16-bit I/O-port space (accesses ports FFFFH and 0000H), the I/O instruction causes a VM exit (the “unconditional I/O exiting” VM-execution control is ignored if the “use I/O bitmaps” VM-execution control is 1).

1. An execution of GETSEC in VMX non-root operation causes a VM exit if CR4.SMXE[Bit 14] = 1 regardless of the value of CPL or RAX. An execution of GETSEC causes an invalid-opcode exception (#UD) if CR4.SMXE[Bit 14] = 0.

2. Under the dual-monitor treatment of SMIs and SMM, executions of VMCALL cause SMM VM exits in VMX root operation outside SMM. See Section 34.15.2.

3. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 27.6.2.

See Section 28.1.1 for information regarding the priority of VM exits relative to faults that may be caused by the INS and OUTS instructions.

- **INVLPG.** The INVLPG instruction causes a VM exit if the “INVLPG exiting” VM-execution control is 1.
- **INVPCID.** The INVPCID instruction causes a VM exit if the “INVLPG exiting” and “enable INVPCID” VM-execution controls are both 1.
- **LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR.** These instructions cause VM exits if the “descriptor-table exiting” VM-execution control is 1.
- **LMSW.** In general, the LMSW instruction causes a VM exit if it would write, for any bit set in the low 4 bits of the CR0 guest/host mask, a value different than the corresponding bit in the CR0 read shadow. LMSW never clears bit 0 of CR0 (CR0.PE); thus, LMSW causes a VM exit if either of the following are true:
 - The bits in position 0 (corresponding to CR0.PE) are set in both the CR0 guest/host mask and the source operand, and the bit in position 0 is clear in the CR0 read shadow.
 - For any bit position in the range 3:1, the bit in that position is set in the CR0 guest/host mask and the values of the corresponding bits in the source operand and the CR0 read shadow differ.
- **LOADIWKEY.** The LOADIWKEY instruction causes a VM exit if the “LOADIWKEY exiting” VM-execution control is 1.
- **MONITOR.** The MONITOR instruction causes a VM exit if the “MONITOR exiting” VM-execution control is 1.
- **MOV from CR3.** The MOV from CR3 instruction causes a VM exit if the “CR3-store exiting” VM-execution control is 1. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
- **MOV from CR8.** The MOV from CR8 instruction causes a VM exit if the “CR8-store exiting” VM-execution control is 1.
- **MOV to CR0.** The MOV to CR0 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR0 guest/host mask, the corresponding bit in the CR0 read shadow. (If every bit is clear in the CR0 guest/host mask, MOV to CR0 cannot cause a VM exit.)
- **MOV to CR3.** The MOV to CR3 instruction causes a VM exit unless the “CR3-load exiting” VM-execution control is 0 or the value of its source operand is equal to one of the CR3-target values specified in the VMCS. Only the first n CR3-target values are considered, where n is the CR3-target count. If the “CR3-load exiting” VM-execution control is 1 and the CR3-target count is 0, MOV to CR3 always causes a VM exit.

The first processors to support the virtual-machine extensions supported only the 1-setting of the “CR3-load exiting” VM-execution control. These processors always consult the CR3-target controls to determine whether an execution of MOV to CR3 causes a VM exit.

- **MOV to CR4.** The MOV to CR4 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR4 guest/host mask, the corresponding bit in the CR4 read shadow.
- **MOV to CR8.** The MOV to CR8 instruction causes a VM exit if the “CR8-load exiting” VM-execution control is 1.
- **MOV DR.** The MOV DR instruction causes a VM exit if the “MOV-DR exiting” VM-execution control is 1. Such VM exits represent an exception to the principles identified in Section 28.1.1 in that they take priority over the following: general-protection exceptions based on privilege level; and invalid-opcode exceptions that occur because CR4.DE=1 and the instruction specified access to DR4 or DR5.
- **MWAIT.** The MWAIT instruction causes a VM exit if the “MWAIT exiting” VM-execution control is 1. If this control is 0, the behavior of the MWAIT instruction may be modified (see Section 28.3).
- **PAUSE.** The behavior of each of this instruction depends on CPL and the settings of the “PAUSE exiting” and “PAUSE-loop exiting” VM-execution controls:
 - CPL = 0.
 - If the “PAUSE exiting” and “PAUSE-loop exiting” VM-execution controls are both 0, the PAUSE instruction executes normally.
 - If the “PAUSE exiting” VM-execution control is 1, the PAUSE instruction causes a VM exit (the “PAUSE-loop exiting” VM-execution control is ignored if CPL = 0 and the “PAUSE exiting” VM-execution control is 1).

- If the “PAUSE exiting” VM-execution control is 0 and the “PAUSE-loop exiting” VM-execution control is 1, the following treatment applies.

The processor determines the amount of time between this execution of PAUSE and the previous execution of PAUSE at CPL 0. If this amount of time exceeds the value of the VM-execution control field PLE_Gap, the processor considers this execution to be the first execution of PAUSE in a loop. (It also does so for the first execution of PAUSE at CPL 0 after VM entry.)

Otherwise, the processor determines the amount of time since the most recent execution of PAUSE that was considered to be the first in a loop. If this amount of time exceeds the value of the VM-execution control field PLE_Window, a VM exit occurs.

For purposes of these computations, time is measured based on a counter that runs at the same rate as the timestamp counter (TSC).

— CPL > 0.

- If the “PAUSE exiting” VM-execution control is 0, the PAUSE instruction executes normally.
- If the “PAUSE exiting” VM-execution control is 1, the PAUSE instruction causes a VM exit.

The “PAUSE-loop exiting” VM-execution control is ignored if CPL > 0.

- **PCONFIG.** The PCONFIG instruction causes a VM exit if the “enable PCONFIG” VM-execution control is 1 and one of the following is true:

- The value of EAX is less than 63 and the corresponding bit in the PCONFIG-exiting bitmap is 1 (see Section 27.6.17).
- The value of EAX is greater than or equal to 63 and bit 63 in the PCONFIG-exiting bitmap is 1.

If the “enable PCONFIG” VM-execution control is 1 and neither of the previous items hold, the PCONFIG instruction executes normally.

- **RDMSR.** The RDMSR instruction causes a VM exit if any of the following are true:
 - The “use MSR bitmaps” VM-execution control is 0.
 - The value of ECX is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
 - The value of ECX is in the range 00000000H – 00001FFFH and bit *n* in read bitmap for low MSRs is 1, where *n* is the value of ECX.
 - The value of ECX is in the range C0000000H – C0001FFFH and bit *n* in read bitmap for high MSRs is 1, where *n* is the value of ECX & 00001FFFH.

See Section 27.6.9 for details regarding how these bitmaps are identified.

- **RDMSRLIST.** The RDMSRLIST instruction causes a VM exit if the “enable MSR-list instructions” VM-execution control is 1 and the “use MSR bitmaps” VM-execution control is 0. If both controls are 1, the instruction reads one MSR at a time normally, storing the value read to memory and clearing the corresponding bit in RCX. An attempt to read MSR *X* causes a VM exit if any of the following are true:
 - *X* is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
 - *X* is in the range 00000000H – 00001FFFH and bit *X* in read bitmap for low MSRs is 1.
 - *X* is in the range C0000000H – C0001FFFH and bit *n* in read bitmap for high MSRs is 1, where *n* is the value of *X* & 00001FFFH.

If an attempt to read an MSR causes a VM exit, the corresponding bit in RCX is not cleared, the MSR is not read, and no value is stored to memory.

- **RDPMC.** The RDPMC instruction causes a VM exit if the “RDPMC exiting” VM-execution control is 1.
- **RDRAND.** The RDRAND instruction causes a VM exit if the “RDRAND exiting” VM-execution control is 1.
- **RDSEED.** The RDSEED instruction causes a VM exit if the “RDSEED exiting” VM-execution control is 1.
- **RDTSC.** The RDTSC instruction causes a VM exit if the “RDTSC exiting” VM-execution control is 1.
- **RDTSCP.** The RDTSCP instruction causes a VM exit if the “RDTSC exiting” and “enable RDTSCP” VM-execution controls are both 1.
- **RSM.** The RSM instruction causes a VM exit if executed in system-management mode (SMM).¹

- **TPAUSE.** The TPAUSE instruction causes a VM exit if the “RDTSC exiting” and “enable user wait and pause” VM-execution controls are both 1.
- **UMWAIT.** The UMWAIT instruction causes a VM exit if the “RDTSC exiting” and “enable user wait and pause” VM-execution controls are both 1.
- **VMREAD.** The VMREAD instruction causes a VM exit if any of the following are true:
 - The “VMCS shadowing” VM-execution control is 0.
 - Bits 63:15 (bits 31:15 outside 64-bit mode) of the register source operand are not all 0.
 - Bit n in VMREAD bitmap is 1, where n is the value of bits 14:0 of the register source operand. See Section 27.6.15 for details regarding how the VMREAD bitmap is identified.

If the VMREAD instruction does not cause a VM exit, it reads from the VMCS referenced by the VMCS link pointer. See Chapter 33, “VMREAD—Read Field from Virtual-Machine Control Structure” for details of the operation of the VMREAD instruction.

- **VMWRITE.** The VMWRITE instruction causes a VM exit if any of the following are true:
 - The “VMCS shadowing” VM-execution control is 0.
 - Bits 63:15 (bits 31:15 outside 64-bit mode) of the register source operand are not all 0.
 - Bit n in VMWRITE bitmap is 1, where n is the value of bits 14:0 of the register source operand. See Section 27.6.15 for details regarding how the VMWRITE bitmap is identified.

If the VMWRITE instruction does not cause a VM exit, it writes to the VMCS referenced by the VMCS link pointer. See Chapter 33, “VMWRITE—Write Field to Virtual-Machine Control Structure” for details of the operation of the VMWRITE instruction.

- **WBINVD.** The WBINVD instruction causes a VM exit if the “WBINVD exiting” VM-execution control is 1.
- **WBNOINVD.** The WBNOINVD instruction causes a VM exit if the “WBINVD exiting” VM-execution control is 1.
- **WRMSR, WRMSRNS.** Execution of one of these instructions causes a VM exit if any of the following are true:
 - The “use MSR bitmaps” VM-execution control is 0.
 - The value of ECX is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
 - The value of ECX is in the range 00000000H – 00001FFFH and bit n in write bitmap for low MSRs is 1, where n is the value of ECX.
 - The value of ECX is in the range C0000000H – C0001FFFH and bit n in write bitmap for high MSRs is 1, where n is the value of ECX & 00001FFFH.

See Section 27.6.9 for details regarding how these bitmaps are identified.

- **WRMSRLIST.** The WRMSRLIST instruction causes a VM exit if the “enable MSR-list instructions” VM-execution control is 1 and the “use MSR bitmaps” VM-execution control is 0. In this case, the register operands of the instructions are not used, memory is not read, and no MSRs are modified.

If both controls are 1, the instruction writes one MSR at a time normally, using a value read from memory and clearing the corresponding bit in RCX. An attempt to write MSR X causes a VM exit if any of the following are true:

- X is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
- X is in the range 00000000H – 00001FFFH and bit X in write bitmap for low MSRs is 1.
- X is in the range C0000000H – C0001FFFH and bit n in write bitmap for high MSRs is 1, where n is the value of X & 00001FFFH.

Such a VM exit occurs after the data that would be written to the MSR is read from memory, but the corresponding bit in RCX is not cleared and the MSR is not written. The data that would have been written to the MSR is saved into the MSR-data field of the VMCS (see Section 30.2.5).

1. Execution of the RSM instruction outside SMM causes an invalid-opcode exception regardless of whether the processor is in VMX operation. It also does so in VMX root operation in SMM; see Section 34.15.3.

- **XRSTORS.** The XRSTORS instruction causes a VM exit if the “enable XSAVES/XRSTORS” VM-execution control is 1 and any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap (see Section 27.6.20).
- **XSAVES.** The XSAVES instruction causes a VM exit if the “enable XSAVES/XRSTORS” VM-execution control is 1 and any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap (see Section 27.6.20).

28.2 OTHER CAUSES OF VM EXITS

In addition to VM exits caused by instruction execution, the following events can cause VM exits:¹

- **Exceptions.** Exceptions (faults, traps, and aborts) cause VM exits based on the exception bitmap (see Section 27.6.3). If an exception occurs, its vector (in the range 0–31) is used to select a bit in the exception bitmap. If the bit is 1, a VM exit occurs; if the bit is 0, the exception is delivered normally. This use of the exception bitmap applies also to exceptions generated by the instructions INT1, INT3, INTO, BOUND, UD0, UD1, and UD2, and UDB.²

Page faults (exceptions with vector 14) are specially treated. When a page fault occurs, a processor consults (1) bit 14 of the exception bitmap; (2) the error code produced with the page fault [PFEC]; (3) the page-fault error-code mask field [PFEC_MASK]; and (4) the page-fault error-code match field [PFEC_MATCH]. It checks if PFEC & PFEC_MASK = PFEC_MATCH. If there is equality, the specification of bit 14 in the exception bitmap is followed (for example, a VM exit occurs if that bit is set). If there is inequality, the meaning of that bit is reversed (for example, a VM exit occurs if that bit is clear).

Thus, if software desires VM exits on all page faults, it can set bit 14 in the exception bitmap to 1 and set the page-fault error-code mask and match fields each to 00000000H. If software desires VM exits on no page faults, it can set bit 14 in the exception bitmap to 1, the page-fault error-code mask field to 00000000H, and the page-fault error-code match field to FFFFFFFFH.

- **Triple fault.** A VM exit occurs if the logical processor encounters an exception while attempting to call the double-fault handler and that exception itself does not cause a VM exit due to the exception bitmap. This applies to the case in which the double-fault exception was generated within VMX non-root operation, the case in which the double-fault exception was generated during event injection by VM entry, and to the case in which VM entry is injecting a double-fault exception.
- **External interrupts.** An external interrupt causes a VM exit if the “external-interrupt exiting” VM-execution control is 1 (see Section 32.6 for an exception.) Otherwise, the processor handles the interrupt normally.³ (If a logical processor is in the shutdown state or the wait-for-SIPI state, external interrupts are blocked. The processor does handle the interrupt and no VM exit occurs.)
- **Non-maskable interrupts (NMIs).** An NMI causes a VM exit if the “NMI exiting” VM-execution control is 1. Otherwise, it is delivered using [vector 2](#). (If a logical processor is in the wait-for-SIPI state, NMIs are blocked. The NMI is not delivered and no VM exit occurs.)
- **INIT signals.** INIT signals cause VM exits. A logical processor performs none of the operations normally associated with these events. Such exits do not modify register state or clear pending events as they would outside of VMX operation. (If a logical processor is in the wait-for-SIPI state, INIT signals are blocked. They do not cause VM exits in this case.)
- **Start-up IPIs (SIPIs). SIPIs cause VM exits.** If a logical processor is not in the wait-for-SIPI activity state when a SIPI arrives, no VM exit occurs and the SIPI is discarded. VM exits due to SIPIs do not perform any of the normal operations associated with those events: they do not modify register state as they would outside of

1. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 27.6.2.

2. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with value 1 or 3 for *n*.

3. Normal handling usually means delivery [to software](#), but it could also mean treatment of the interrupt as a user-interrupt notification.

VMX operation. (If a logical processor is not in the wait-for-SIPI state, SIPIs are blocked. They do not cause VM exits in this case.)

- **Task switches.** Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. See Section 28.4.2.
- **System-management interrupts (SMIs).** If the logical processor is using the dual-monitor treatment of SMIs and system-management mode (SMM), SMIs cause SMM VM exits. See Section 34.15.2.¹
- **VMX-preemption timer.** A VM exit occurs when the timer counts down to zero. See Section 28.5.1 for details of operation of the VMX-preemption timer.

Debug-trap exceptions and higher priority events take priority over VM exits caused by the VMX-preemption timer. VM exits caused by the VMX-preemption timer take priority over VM exits caused by the “NMI-window exiting” VM-execution control and lower priority events.

These VM exits wake a logical processor from the same inactive states as would a non-maskable interrupt. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

- **Bus locks.** Assertion of a bus lock (see Section 11.1.2) causes a VM exit if the “VMM bus-lock detection” VM-execution control is 1. Such a VM exit is trap-like because it is generated after execution of an instruction that asserts a bus lock. The VM exit thus does not prevent assertion of the bus lock. These VM exits take priority over system-management interrupts (SMIs), INIT signals, and lower priority events.
- **Instruction timeout.** If the “instruction timeout” VM-execution control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within the amount of time specified by the instruction-timeout control VM-execution control field (see Section 27.6.24).

In addition, there are controls that cause VM exits based on the readiness of guest software to receive interrupts:

- If the “interrupt-window exiting” VM-execution control is 1, a VM exit occurs before execution of any instruction if RFLAGS.IF = 1 and there is no blocking of events by STI or by MOV SS (see Table 27-3).

Non-maskable interrupts (NMIs) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over external interrupts and lower priority events.

These VM exits wake a logical processor from the same inactive states as would an external interrupt. Specifically, they wake a logical processor from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the shutdown state or the wait-for-SIPI state.

- If the “NMI-window exiting” VM-execution control is 1, a VM exit occurs before execution of any instruction if there is no virtual-NMI blocking and there is no blocking of events by MOV SS and no blocking of events by STI (see Table 27-3).

VM exits caused by the VMX-preemption timer and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.

These VM exits wake a logical processor from the same inactive states as would an NMI. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

Conditions necessary for some of the VM exits identified in this section may hold immediately after VM entry. If they do, a corresponding VM exit occurs at that time.

28.3 CHANGES TO INSTRUCTION BEHAVIOR IN VMX NON-ROOT OPERATION

The behavior of some instructions is changed in VMX non-root operation. Some of these changes are determined by the settings of certain VM-execution control fields. The following items detail such changes:²

- **CLTS.** Behavior of the CLTS instruction is determined by the bits in position 3 (corresponding to CR0.TS) in the CR0 guest/host mask and the CR0 read shadow:

1. Under the dual-monitor treatment of SMIs and SMM, SMIs also cause SMM VM exits if they occur in VMX root operation outside SMM. If the processor is using the default treatment of SMIs and SMM, SMIs are delivered as described in Section 34.14.1.

- If bit 3 in the CR0 guest/host mask is 0, CLTS clears CR0.TS normally (the value of bit 3 in the CR0 read shadow is irrelevant in this case), unless CR0.TS is fixed to 1 in VMX operation (see Section 26.8), in which case CLTS causes a general-protection exception.
- If bit 3 in the CR0 guest/host mask is 1 and bit 3 in the CR0 read shadow is 0, CLTS completes but does not change the contents of CR0.TS.
- If the bits in position 3 in the CR0 guest/host mask and the CR0 read shadow are both 1, CLTS causes a VM exit.

- **ENQCMD, ENQCMLS.** Each of these instructions performs a 64-byte enqueue store that includes a PASID value in bits 19:0. For ENQCMD, the PASID is normally the value of IA32_PASID[19:0], while for ENQCMLS, the PASID is normally read from memory.

The behavior of each of these instructions (and in particular the PASID value used for the enqueue store) is determined by the setting of the “PASID translation” VM-execution control:

- If the “PASID translation” VM-execution control is 0, the instruction operates normally.
- If the “PASID translation” VM-execution control is 1, the PASID value used for the enqueue store is determined by the PASID-translation process described in Section 28.5.8. (Note the PASID translation may result in a VM exit, in which case the enqueue store is not performed.)

An execution of ENQCMD or ENQCMLS performs PASID translation only after checking for conditions that may result in general-protection exception (the check of IA32_PASID.Valid for ENQCMD; the privilege-level check for ENQCMLS), after loading the instruction's source operand from memory, and thus after any faults or VM exits that the loading may cause (e.g., page faults or EPT violations). PASID translation occurs before the actual enqueue store and thus before any faults or VM exits that it may cause.

- **ERETS, ERETU.** Each of these instructions unblocks NMIs if bit 18 is set in the augmented SS in the return state on the stack. The following items detail how this behavior may be changed in VMX non-root operation, depending on the settings of certain VM-execution controls:
 - If the “NMI exiting” VM-execution control is 0, this behavior of ERETU and ERETU is not modified (they unblock NMIs as indicated above). (If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0; see Section 29.2.1.1.)
 - If the “NMI exiting” VM-execution control is 1, ERETU and ERETU do not unblock physical NMIs, regardless of the return state on the stack. If, in addition, the “virtual NMIs” VM-execution control is 1, the logical processor tracks virtual-NMI blocking. In this case, ERETU and ERETU each unblocks virtual NMIs if bit 18 is set in the augmented SS in the return state.

If the “NMI exiting” VM-execution control is 1 and the “virtual NMIs” VM-execution control is 0, ERETU and ERETU ignore bit 18 of the augmented SS.

- **INVPCID.** Behavior of the INVPCID instruction is determined first by the setting of the “enable INVPCID” VM-execution control:
 - If the “enable INVPCID” VM-execution control is 0, INVPCID causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable INVPCID” VM-execution control is 1, treatment is based on the setting of the “INVLPG exiting” VM-execution control:
 - If the “INVLPG exiting” VM-execution control is 0, INVPCID operates normally.
 - If the “INVLPG exiting” VM-execution control is 1, INVPCID causes a VM exit.
- **IRET.** Behavior of IRET with regard to NMI blocking (see Table 27-3) is determined by the settings of the “NMI exiting” and “virtual NMIs” VM-execution controls:
 - If the “NMI exiting” VM-execution control is 0, IRET operates normally and unblocks NMIs. (If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0; see Section 29.2.1.1.)

-
2. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 27.6.2.

- If the “NMI exiting” VM-execution control is 1, IRET does not affect blocking of NMIs. If, in addition, the “virtual NMIs” VM-execution control is 1, the logical processor tracks virtual-NMI blocking. In this case, IRET removes any virtual-NMI blocking.

The unblocking of NMIs or virtual NMIs specified above occurs even if IRET causes a fault.

- **LMSW.** Outside of VMX non-root operation, LMSW loads its source operand into CR0[3:0], but it does not clear CR0.PE if that bit is set. In VMX non-root operation, an execution of LMSW that does not cause a VM exit (see Section 28.1.3) leaves unmodified any bit in CR0[3:0] corresponding to a bit set in the CR0 guest/host mask. An attempt to set any other bit in CR0[3:0] to a value not supported in VMX operation (see Section 26.8) causes a general-protection exception. Attempts to clear CR0.PE are ignored without fault.

- **MOV from CR0.** The behavior of MOV from CR0 is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, MOV from CR0 reads normally from CR0; if every bit is set in the CR0 guest/host mask, MOV from CR0 returns the value of the CR0 read shadow.

Depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.

- **MOV from CR3.** If the “enable EPT” VM-execution control is 1 and an execution of MOV from CR3 does not cause a VM exit (see Section 28.1.3), the value loaded from CR3 is a guest-physical address; see Section 31.3.1.
- **MOV from CR4.** The behavior of MOV from CR4 is determined by the CR4 guest/host mask and the CR4 read shadow. For each position corresponding to a bit clear in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR4. For each position corresponding to a bit set in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR4 read shadow. Thus, if every bit is cleared in the CR4 guest/host mask, MOV from CR4 reads normally from CR4; if every bit is set in the CR4 guest/host mask, MOV from CR4 returns the value of the CR4 read shadow.

Depending on the contents of the CR4 guest/host mask and the CR4 read shadow, bits may be set in the destination that would never be set when reading directly from CR4.

- **MOV from CR8.** If the MOV from CR8 instruction does not cause a VM exit (see Section 28.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 32.3.
- **MOV to CR0.** An execution of MOV to CR0 that does not cause a VM exit (see Section 28.1.3) leaves unmodified any bit in CR0 corresponding to a bit set in the CR0 guest/host mask. Treatment of attempts to modify other bits in CR0 depends on the setting of the “unrestricted guest” VM-execution control:
 - If the control is 0, MOV to CR0 causes a general-protection exception if it attempts to set any bit in CR0 to a value not supported in VMX operation (see Section 26.8).
 - If the control is 1, MOV to CR0 causes a general-protection exception if it attempts to set any bit in CR0 other than bit 0 (PE) or bit 31 (PG) to a value not supported in VMX operation. It remains the case, however, that MOV to CR0 causes a general-protection exception if it would result in CR0.PE = 0 and CR0.PG = 1 or if it would result in CR0.PG = 1, CR4.PAE = 0, and IA32_EFER.LME = 1.
- **MOV to CR3.** If the “enable EPT” VM-execution control is 1 and an execution of MOV to CR3 does not cause a VM exit (see Section 28.1.3), the value loaded into CR3 is treated as a guest-physical address; see Section 31.3.1. The following details apply:
 - An attempt to set a reserved bit in CR3[63:M] results in #GP(0), where M is the value enumerated in CPUID.80000008H:EAX[7:0]. (Unlike other cases, the value M is not reduced outside SEAM.)
 - If PAE paging is not being used, the instruction does not use the guest-physical address to access memory and it does not cause it to be translated through EPT.¹
 - If PAE paging is being used, the instruction translates the guest-physical address through EPT and uses the result to load the four (4) page-directory-pointer-table entries (PDPTes). The instruction does not use the guest-physical addresses the PDPTes to access memory and it does not cause them to be translated

1. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

through EPT. An attempt to set a PDPTE bit in positions 63:M results in #GP(0), where M is defined above (M is not reduced outside SEAM).

- **MOV to CR4.** An execution of MOV to CR4 that does not cause a VM exit (see Section 28.1.3) leaves unmodified any bit in CR4 corresponding to a bit set in the CR4 guest/host mask. Such an execution causes a general-protection exception if it attempts to set any bit in CR4 (not corresponding to a bit set in the CR4 guest/host mask) to a value not supported in VMX operation (see Section 26.8).
- **MOV to CR8.** If the MOV to CR8 instruction does not cause a VM exit (see Section 28.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 32.3.
- **MWAIT.** Behavior of the MWAIT instruction (which always causes an invalid-opcode exception—#UD—if CPL > 0) is determined by the setting of the “MWAIT exiting” VM-execution control:
 - If the “MWAIT exiting” VM-execution control is 1, MWAIT causes a VM exit.
 - If the “MWAIT exiting” VM-execution control is 0, MWAIT operates normally if one of the following are true: (1) ECX[0] is 0; (2) RFLAGS.IF = 1; or both of the following are true: (a) the “interrupt-window exiting” VM-execution control is 0; and (b) the logical processor has not recognized a pending virtual interrupt (see Section 29.2.1).
 - If the “MWAIT exiting” VM-execution control is 0, ECX[0] = 1, and RFLAGS.IF = 0, MWAIT does not cause the processor to enter an implementation-dependent optimized state if either the “interrupt-window exiting” VM-execution control is 1 or the logical processor has recognized a pending virtual interrupt; instead, control passes to the instruction following the MWAIT instruction.
- **PBNDKB.** Behavior of the PBNDKB instruction is determined by the setting of the “enable PBNDKB” VM-execution control:
 - If the “enable PBNDKB” VM-execution control is 0, PBNDKB causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable PBNDKB” VM-execution control is 1, PBNDKB operates normally.
- **PCONFIG.** Behavior of the PCONFIG instruction is determined by the setting of the “enable PCONFIG” VM-execution control:
 - If the “enable PCONFIG” VM-execution control is 0, PCONFIG causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable PCONFIG” VM-execution control is 1, PCONFIG may cause a VM exit as specified in Section 28.1.3; if it does not cause such a VM exit, it operates normally.
- **RDMSR.** Section 28.1.3 identifies when executions of the RDMSR instruction cause VM exits. If such an execution causes neither a fault due to CPL > 0 nor a VM exit, the instruction’s behavior may be modified for certain values of ECX:
 - If ECX contains 10H (indicating the IA32_TIME_STAMP_COUNTER MSR), the value returned by the instruction is determined by the setting of the “use TSC offsetting” VM-execution control:
 - If the control is 0, RDMSR operates normally, loading EAX:EDX with the value of the IA32_TIME_STAMP_COUNTER MSR.
 - If the control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDMSR loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
 - If the control is 1, RDMSR first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.

The 1-setting of the “use TSC offsetting” VM-execution control does not affect executions of RDMSR if ECX contains 6E0H (indicating the IA32_TSC_DEADLINE MSR). Such executions return the APIC-timer deadline relative to the actual timestamp counter without regard to the TSC offset.

 - If ECX contains 48H (indicating the IA32_SPEC_CTRL MSR), the value returned by the instruction is determined by the setting of the “virtualize IA32_SPEC_CTRL” VM-execution control:

- If the control is 0, RDMSR operates normally, loading EAX:EDX with the value of the IA32_SPEC_CTRL MSR.
- If the control is 1, the value returned is that of the IA32_SPEC_CTRL shadow field in the VMCS.
- If ECX is in the range 800H–8FFH (indicating an APIC MSR), instruction behavior may be modified if the “virtualize x2APIC mode” VM-execution control is 1; see Section 32.5.
- **RDMSRLIST.** Behavior of the RDMSRLIST instruction is determined first by the setting of the “enable MSR-list instructions” VM-execution control:
 - If the “enable MSR-list instructions” VM-execution control is 0, RDMSRLIST causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable MSR-list instructions” VM-execution control is 1, the instruction causes a general-protection exception (#GP) normally if CPL > 0. Otherwise, its operation depends on the setting of the “use MSR bitmaps” VM-execution control:
 - If the control is 0, the instruction causes a VM exit.
 - If the control is 1, the instruction commences normally, reading one MSR at a time. Reads of certain MSRs are treated specially as described above for RDMSR. In addition, attempts to access specific MSRs may cause VM exits; see Section 28.1.3 for details.
- **RDPID.** Behavior of the RDPID instruction is determined first by the setting of the “enable RDTSCP” VM-execution control:
 - If the “enable RDTSCP” VM-execution control is 0, RDPID causes an invalid-opcode exception (#UD).
 - If the “enable RDTSCP” VM-execution control is 1, RDPID operates normally.
- **RDTS.** Behavior of the RDTS instruction is determined by the settings of the “RDTS exiting” and “use TSC offsetting” VM-execution controls:
 - If both controls are 0, RDTS operates normally.
 - If the “RDTS exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDTS loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
 - If the control is 1, RDTS first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.
 - If the “RDTS exiting” VM-execution control is 1, RDTS causes a VM exit.
- **RDTSCP.** Behavior of the RDTSCP instruction is determined first by the setting of the “enable RDTSCP” VM-execution control:
 - If the “enable RDTSCP” VM-execution control is 0, RDTSCP causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable RDTSCP” VM-execution control is 1, treatment is based on the settings of the “RDTS exiting” and “use TSC offsetting” VM-execution controls:
 - If both controls are 0, RDTSCP operates normally.
 - If the “RDTS exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDTSCP loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
 - If the control is 1, RDTSCP first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.

In either case, RDTSCP also loads ECX with the value of bits 31:0 of the IA32_TSC_AUX MSR.

- If the “RDTS exiting” VM-execution control is 1, RDTSCP causes a VM exit.

- **SMSW.** The behavior of SMSW is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, SMSW reads normally from CR0; if every bit is set in the CR0 guest/host mask, SMSW returns the value of the CR0 read shadow.

Note the following: (1) for any memory destination or for a 16-bit register destination, only the low 16 bits of the CR0 guest/host mask and the CR0 read shadow are used (bits 63:16 of a register destination are left unchanged); (2) for a 32-bit register destination, only the low 32 bits of the CR0 guest/host mask and the CR0 read shadow are used (bits 63:32 of the destination are cleared); and (3) depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.

- **TPAUSE.** Behavior of the TPAUSE instruction is determined first by the setting of the “enable user wait and pause” VM-execution control:
 - If the “enable user wait and pause” VM-execution control is 0, TPAUSE causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable user wait and pause” VM-execution control is 1, treatment is based on the setting of the “RDTSC exiting” VM-execution control:
 - If the “RDTSC exiting” VM-execution control is 0, the instruction delays for an amount of time called here the **physical delay**. The physical delay is first computed by determining the **virtual delay** (the time to delay relative to the guest’s timestamp counter).
 If IA32_UWAIT_CONTROL[31:2] is zero, the virtual delay is the value in EDX:EAX minus the value that RDTSC would return (see above); if IA32_UWAIT_CONTROL[31:2] is not zero, the virtual delay is the minimum of that difference and AND(IA32_UWAIT_CONTROL, FFFFFFFCH).
 The physical delay depends upon the settings of the “use TSC offsetting” and “use TSC scaling” VM-execution controls:
 - If either control is 0, the physical delay is the virtual delay.
 - If both controls are 1, the virtual delay is multiplied by 2^{48} (using a shift) to produce a 128-bit integer. That product is then divided by the TSC multiplier to produce a 64-bit integer. The physical delay is that quotient.
 - If the “RDTSC exiting” VM-execution control is 1, TPAUSE causes a VM exit.

- **UMONITOR.** Behavior of the UMONITOR instruction is determined by the setting of the “enable user wait and pause” VM-execution control:

- If the “enable user wait and pause” VM-execution control is 0, UMONITOR causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
- If the “enable user wait and pause” VM-execution control is 1, UMONITOR operates normally.

- **UWAIT.** Behavior of the UWAIT instruction is determined first by the setting of the “enable user wait and pause” VM-execution control:

- If the “enable user wait and pause” VM-execution control is 0, UWAIT causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
- If the “enable user wait and pause” VM-execution control is 1, treatment is based on the setting of the “RDTSC exiting” VM-execution control:

- If the “RDTSC exiting” VM-execution control is 0, and if the instruction causes a delay, the amount of time delayed is called here the **physical delay**. The physical delay is first computed by determining the **virtual delay** (the time to delay relative to the guest’s timestamp counter).

If IA32_UWAIT_CONTROL[31:2] is zero, the virtual delay is the value in EDX:EAX minus the value that RDTSC would return (see above); if IA32_UWAIT_CONTROL[31:2] is not zero, the virtual delay is the minimum of that difference and AND(IA32_UWAIT_CONTROL, FFFFFFFCH).

The physical delay depends upon the settings of the “use TSC offsetting” and “use TSC scaling” VM-execution controls:

- If either control is 0, the physical delay is the virtual delay.
 - If both controls are 1, the virtual delay is multiplied by 2^{48} (using a shift) to produce a 128-bit integer. That product is then divided by the TSC multiplier to produce a 64-bit integer. The physical delay is that quotient.
- If the “RDTSC exiting” VM-execution control is 1, UMWAIT causes a VM exit.
- **WRMSR, WRMSRNS.** Section 28.1.3 identifies when an execution of WRMSR or WRMSRNS would cause a VM exit. If such an execution causes neither a fault due to CPL > 0 nor a VM exit, the instruction’s behavior may be modified for certain values of ECX:
 - If ECX contains 48H (indicating the IA32_SPEC_CTRL MSR), instruction behavior depends on the setting of the “virtualize IA32_SPEC_CTRL” VM-execution control:
 - If the control is 0, WRMSR and WRMSRNS operate normally, loading the IA32_SPEC_CTRL MSR with the value in EAX:EDX.
 - If the control is 1, the instruction will attempt to write the IA32_SPEC_CTRL MSR using the instruction’s source operand, but it will attempt to modify only those bits in positions corresponding to bits cleared in the IA32_SPEC_CTRL mask field in the VMCS.

Specifically, the instruction attempts to write the MSR with the following value:

$$(MSR\_VAL \& ISC\_MASK) \text{ OR } (SRC \& \text{NOT } ISC\_MASK),$$

where MSR_VAL is the original value of the MSR, ISC_MASK is the IA32_SPEC_CTRL mask, and SRC is the instruction’s source operand.

Any fault that would result from writing that value to the MSR (e.g., due to a reserved-bit violation) occurs normally. Otherwise, the value is written to the MSR.

Such a write to the MSR will have any side effects that would occur normally had the MSR been written with the value indicated above (including any side effects that may result from writing unchanged values to the masked bits).

If the write completes without a fault, the unmodified value of the source operand is written to the IA32_SPEC_CTRL shadow field in the VMCS.
 - If ECX contains 79H (indicating IA32_BIOS_UPDT_TRIG MSR), no microcode update is loaded, and control passes to the next instruction. This implies that microcode updates cannot be loaded in VMX non-root operation.
 - On processors that support Intel PT but which do not allow it to be used in VMX operation, if ECX contains 570H (indicating the IA32_RTIT_CTL MSR), the instruction causes a general-protection exception.¹
 - If ECX contains 808H (indicating the TPR MSR), 80BH (the EOI MSR), 830H (the ICR MSR), or 83FH (the self-IPI MSR), instruction behavior may be modified if the “virtualize x2APIC mode” VM-execution control is 1; see Section 32.5.
- **WRMSRLIST.** Behavior of the WRMSRLIST instruction is determined first by the setting of the “enable MSR-list instructions” VM-execution control:
 - If the “enable MSR-list instructions” VM-execution control is 0, WRMSRLIST causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable MSR-list instructions” VM-execution control is 1, the instruction causes a general-protection exception (#GP) normally if CPL > 0. Otherwise, its operation depends on the setting of the “use MSR bitmaps” VM-execution control:
 - If the control is 0, the instruction causes a VM exit.
 - If the control is 1, the instruction commences normally, writing one MSR at a time. Writes to certain MSRs are treated specially as described above for WRMSR and WRMSRNS. In addition, attempts to access specific MSRs may cause VM exits; see Section 28.1.3 for details.

1. Software should read the VMX capability MSR IA32_VMX_MISC to determine whether the processor allows Intel PT to be used in VMX operation (see Appendix A.6).

- **XRSTORS.** Behavior of the XRSTORS instruction is determined first by the setting of the “enable XSAVES/XRSTORS” VM-execution control:
 - If the “enable XSAVES/XRSTORS” VM-execution control is 0, XRSTORS causes an invalid-opcode exception (#UD).
 - If the “enable XSAVES/XRSTORS” VM-execution control is 1, treatment is based on the value of the XSS-exiting bitmap (see Section 27.6.20):
 - XRSTORS causes a VM exit if any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
 - Otherwise, XRSTORS operates normally.
- **XSAVES.** Behavior of the XSAVES instruction is determined first by the setting of the “enable XSAVES/XRSTORS” VM-execution control:
 - If the “enable XSAVES/XRSTORS” VM-execution control is 0, XSAVES causes an invalid-opcode exception (#UD).
 - If the “enable XSAVES/XRSTORS” VM-execution control is 1, treatment is based on the value of the XSS-exiting bitmap (see Section 27.6.20):
 - XSAVES causes a VM exit if any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
 - Otherwise, XSAVES operates normally.

28.4 OTHER CHANGES IN VMX NON-ROOT OPERATION

Treatments of event blocking, task switches, and certain shadow-stack updates may differ in VMX non-root operation as described in the following sections.

28.4.1 Event Blocking

Event blocking is modified in VMX non-root operation as follows:

- If the “external-interrupt exiting” VM-execution control is 1, RFLAGS.IF does not control the blocking of external interrupts. In this case, an external interrupt that is not blocked for other reasons causes a VM exit (even if RFLAGS.IF = 0).
- If the “external-interrupt exiting” VM-execution control is 1, external interrupts may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).
- If the “NMI exiting” VM-execution control is 1, non-maskable interrupts (NMIs) may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).

28.4.2 Treatment of Task Switches

Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. However, the following checks are performed (in the order indicated), possibly resulting in a fault, before there is any possibility of a VM exit due to task switch:

1. If a task gate is being used, appropriate checks are made on its P bit and on the proper values of the relevant privilege fields. The following cases detail the privilege checks performed:
 - a. If CALL, INT *n*, INT1, INT3, INTO, or JMP accesses a task gate in IA-32e mode, a general-protection exception occurs.
 - b. If CALL, INT *n*, INT3, INTO, or JMP accesses a task gate outside IA-32e mode, privilege-levels checks are performed on the task gate but, if they pass, privilege levels are not checked on the referenced task-state segment (TSS) descriptor.
 - c. If CALL or JMP accesses a TSS descriptor directly in IA-32e mode, a general-protection exception occurs.

- d. If CALL or JMP accesses a TSS descriptor directly outside IA-32e mode, privilege levels are checked on the TSS descriptor.
 - e. If a non-maskable interrupt (NMI), an exception, or an external interrupt accesses a task gate in the IDT in IA-32e mode, a general-protection exception occurs.
 - f. If a non-maskable interrupt (NMI), an exception other than breakpoint exceptions (#BP) and overflow exceptions (#OF), or an external interrupt accesses a task gate in the IDT outside IA-32e mode, no privilege checks are performed.
 - g. If IRET is executed with RFLAGS.NT = 1 in IA-32e mode, a general-protection exception occurs.
 - h. If IRET is executed with RFLAGS.NT = 1 outside IA-32e mode, a TSS descriptor is accessed directly and no privilege checks are made.
2. Checks are made on the new TSS selector (for example, that is within GDT limits).
 3. The new TSS descriptor is read. (A page fault results if a relevant GDT page is not present).
 4. The TSS descriptor is checked for proper values of type (depends on type of task switch), P bit, S bit, and limit.

Only if checks 1–4 all pass (do not generate faults) might a VM exit occur. However, the ordering between a VM exit due to a task switch and a page fault resulting from accessing the old TSS or the new TSS is implementation-specific. Some processors may generate a page fault (instead of a VM exit due to a task switch) if accessing either TSS would cause a page fault. Other processors may generate a VM exit due to a task switch even if accessing either TSS would cause a page fault.

If an attempt at a task switch through a task gate in the IDT causes an exception (before generating a VM exit due to the task switch) and that exception causes a VM exit, information about the event whose delivery that accessed the task gate is recorded in the [original-event](#) fields and information about the exception that caused the VM exit is recorded in the [exiting-event](#) fields. See Section 30.2. The fact that a task gate was being accessed is not recorded in the VMCS.

If an attempt at a task switch through a task gate in the IDT causes VM exit due to the task switch, information about the event whose delivery accessed the task gate is recorded in the [original-event](#) fields of the VMCS. Since the cause of such a VM exit is a task switch and not [some other event](#), the valid bit for the [exiting-event identification](#) field is 0. See Section 30.2.

28.4.3 Shadow-Stack Updates

As noted in Section 18.2.3, “Supervisor Shadow Stack Token,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, a switch of shadow stack may occur as part of IDT event delivery or an execution of far CALL that changes the CPL, or as part of IDT event delivery that uses the interrupt stack table (IST).

As part of the shadow-stack switch, the processor gains exclusive access to the new stack through manipulation of the **supervisor shadow stack token** located at the base of the new shadow stack. The processor reads the token and, among other things, confirms that bit 0 of the token (its **busy bit**) is 0. If the busy bit is already 1, the transition (event delivery or far CALL) cause a general-protection fault and does not complete. If the busy bit is 0, the transition sets the busy bit by writing to the token in memory. (The update is atomic with the original read of the token.)

If the transition commenced with CPL < 3, it will follow the token update by pushing three items on the new shadow stack (for the old values of the CS selector, instruction pointer, and shadow-stack pointer). As noted in Section 18.2.3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, if any of the pushes causes a VM exit, the processor will revert to the old shadow stack and the busy bit in the new shadow stack’s token remains set. The new shadow stack is said to be prematurely busy.

If the “prematurely busy shadow stack” VM-exit control is 1, a VM exit that results in a shadow stack becoming prematurely busy will indicate that fact through information saved in the VMCS. See Section 30.2.1.

28.5 FEATURES SPECIFIC TO VMX NON-ROOT OPERATION

Some VM-execution controls support features that are specific to VMX non-root operation. These are the VMX-preemption timer (Section 28.5.1) and the monitor trap flag (Section 28.5.2), translation of guest-physical addresses (Section 28.5.3 and Section 28.5.4), APIC virtualization (Section 28.5.5), VM functions (Section 28.5.6), and virtualization exceptions (Section 28.5.7).

28.5.1 VMX-Preemption Timer

If the last VM entry was performed with the 1-setting of “activate VMX-preemption timer” VM-execution control, the **VMX-preemption timer** counts down (from the value loaded by VM entry; see Section 29.7.4) in VMX non-root operation. When the timer counts down to zero, it stops counting down and a VM exit occurs (see Section 28.2).

The VMX-preemption timer counts down at rate proportional to that of the timestamp counter (TSC). Specifically, the timer counts down by 1 every time bit X in the TSC changes due to a TSC increment. The value of X is in the range 0–31 and can be determined by consulting the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

The VMX-preemption timer operates in the C-states C0, C1, and C2; it also operates in the shutdown and wait-for-SIPI states. If the timer counts down to zero in any state other than the wait-for SIPI state, the logical processor transitions to the C0 C-state and causes a VM exit; the timer does not cause a VM exit if it counts down to zero in the wait-for-SIPI state. The timer is not decremented in C-states deeper than C2.

Treatment of the timer in the case of system management interrupts (SMIs) and system-management mode (SMM) depends on whether the treatment of SMIs and SMM:

- If the default treatment of SMIs and SMM (see Section 34.14) is active, the VMX-preemption timer counts across an SMI to VMX non-root operation, subsequent execution in SMM, and the return from SMM via the RSM instruction. However, the timer can cause a VM exit only from VMX non-root operation. If the timer expires during SMI, in SMM, or during RSM, a timer-induced VM exit occurs immediately after RSM with its normal priority unless it is blocked based on activity state (Section 28.2).
- If the dual-monitor treatment of SMIs and SMM (see Section 34.15) is active, transitions into and out of SMM are VM exits and VM entries, respectively. The treatment of the VMX-preemption timer by those transitions is mostly the same as for ordinary VM exits and VM entries; Section 34.15.2 and Section 34.15.4 detail some differences.

28.5.2 Monitor Trap Flag

The **monitor trap flag** is a debugging feature that causes VM exits to occur on certain instruction boundaries in VMX non-root operation. Such VM exits are called **MTF VM exits**. An MTF VM exit may occur on an instruction boundary in VMX non-root operation as follows:

- If the “monitor trap flag” VM-execution control is 1 and VM entry is injecting a vectored event (see Section 29.6.1), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry.
- If VM entry is injecting a pending MTF VM exit (see Section 29.6.2), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0.
- If the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and a pending event (e.g., debug exception or interrupt) is delivered before an instruction can execute, an MTF VM exit is pending on the instruction boundary following delivery of the event (or any nested exception).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is a REP-prefixed string instruction:
 - If the first iteration of the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).
 - If the first iteration of the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary after that iteration.

- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is the XBEGIN instruction. In this case, an MTF VM exit is pending at the fallback instruction address of the XBEGIN instruction. This behavior applies regardless of whether advanced debugging of RTM transactional regions has been enabled (see Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is neither a REP-prefixed string instruction or the XBEGIN instruction:
 - If the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).¹
 - If the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary following execution of that instruction. If the instruction is INT1, INT3, or INTO, this boundary follows delivery of any software exception. If the instruction is INT *n*, this boundary follows delivery of a software interrupt. If the instruction is HLT, the MTF VM exit will be from the HLT activity state.

No MTF VM exit occurs if another VM exit occurs before reaching the instruction boundary on which an MTF VM exit would be pending (e.g., due to an exception or triple fault).

An MTF VM exit occurs on the instruction boundary on which it is pending unless a higher priority event takes precedence or the MTF VM exit is blocked due to the activity state:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over MTF VM exits. MTF VM exits take priority over debug-trap exceptions and lower priority events.
- No MTF VM exit occurs if the processor is in either the shutdown activity state or wait-for-SIPI activity state. If a non-maskable interrupt subsequently takes the logical processor out of the shutdown activity state without causing a VM exit, an MTF VM exit is pending after delivery of that interrupt.

Special treatment may apply to Intel SGX instructions or if the logical processor is in enclave mode. See Section 43.2 for details.

28.5.3 Translation of Guest-Physical Addresses Using EPT

The extended page-table mechanism (EPT) is a feature that can be used to support the virtualization of physical memory. When EPT is in use, certain physical addresses are treated as guest-physical addresses and are not used to access memory directly. Instead, guest-physical addresses are translated by traversing a set of EPT paging structures to produce physical addresses that are used to access memory.

Details of the EPT mechanism are given in Section 31.3.

28.5.4 Translation of Guest-Physical Addresses Used by Intel Processor Trace

As described in Chapter 36, Intel® Processor Trace (Intel PT) captures information about software execution using dedicated hardware facilities.

Intel PT can be configured so that the trace output is written to memory using physical addresses. For example, when the ToPA (table of physical addresses) output mechanism is used, the IA32_RTIT_OUTPUT_BASE MSR contains the physical address of the base of the current ToPA. Each entry in that table contains the physical address of an output region in memory. When an output region becomes full, the ToPA output mechanism directs subsequent trace output to the next output region as indicated in the ToPA.

When the “Intel PT uses guest physical addresses” VM-execution control is 1, the logical processor treats the addresses used by Intel PT (the output addresses as well as those used to discover the output addresses) as guest-physical addresses, translating to physical addresses using EPT before trace output is written to memory.

Translating these addresses through EPT implies that the trace-output mechanism may cause EPT violations and VM exits; details are provided in Section 28.5.4.1. Section 28.5.4.2 describes a mechanism that ensures that these VM exits do not cause loss of trace data.

1. This item includes the cases of an invalid opcode exception—#UD—generated by the UD0, UD1, UD2, and UDB instructions and a BOUND-range exceeded exception—#BR—generated by the BOUND instruction.

28.5.4.1 Guest-Physical Address Translation for Intel PT: Details

When the “Intel PT uses guest physical addresses” VM-execution control is 1, the addresses used by Intel PT are treated as guest-physical addresses and translated using EPT. These addresses include the addresses of the output regions as well as the addresses of the ToPA entries that contain the output-region addresses.

Translation of accesses by the trace-output process may result in EPT violations or EPT misconfigurations (Section 31.3.3), resulting in VM exits. EPT violations resulting for the trace-output process always cause VM exits and are never converted to virtualization exceptions (Section 28.5.7.1).

If no EPT violation or EPT misconfiguration occurs and if page-modification logging (Section 31.3.6) is enabled, the address of an output region may be added to the page-modification log. If the log is full, a page-modification log-full event occurs, resulting in a VM exit.

If the “virtualize APIC accesses” VM-execution control is 1, a guest-physical address used by the trace-output process may be translated to an address on the APIC-access page. In this case, the access by the trace-output process causes an APIC-access VM exit as discussed in Section 32.4.6.1.

28.5.4.2 Trace-Address Pre-Translation (TAPT)

Because it buffers trace data produced by Intel PT before it is written to memory, the processor ensures that buffered data is not lost when a VM exit disables Intel PT. Specifically, the processor ensures that there is sufficient space left in the current output page for the buffered data. If this were not done, buffered trace data could be lost and the resulting trace corrupted.

To prevent the loss of buffered trace data, the processor uses a mechanism called **trace-address pre-translation (TAPT)**. With TAPT, the processor translates using EPT the guest-physical address of the current output region before that address would be used to write buffered trace data to memory.

Because of TAPT, no translation (and thus no EPT violation) occurs at the time output is written to memory; the writes to memory use translations that were cached as part of TAPT. (The details given in Section 28.5.4.1 apply to TAPT.) TAPT ensures that, if a write to the output region would cause an EPT violation, the resulting VM exit is delivered at the time of TAPT, before the region would be used. This allows software to resolve the EPT violation at that time and ensures that, when it is necessary to write buffered trace data to memory, that data will not be lost due to an EPT violation.

TAPT (and resulting VM exits) may occur at any of the following times:

- When software in VMX non-root operation enables tracing by loading the IA32_RTIT_CTL MSR to set the TraceEn bit, using the WRMSR instruction or the XRSTORS instruction.
Any VM exit resulting from TAPT in this case is trap-like: the WRMSR or XRSTORS completes before the VM exit occurs (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).
- At an instruction boundary when one output region becomes full and Intel PT transitions to the next output region.
VM exits resulting from TAPT in this case take priority over any pending debug exceptions. Such a VM exit will save information about such exceptions in the guest-state area of the VMCS.
- As part of a VM entry that enables Intel PT. See Section 29.5 for details.

TAPT may translate not only the guest-physical address of the current output region but those of subsequent output regions as well. (Doing so may provide better protection of trace data.) This implies that any VM exits resulting from TAPT may result from the translation of output-region addresses other than that of the current output region.

28.5.5 APIC Virtualization

APIC virtualization is a collection of features that can be used to support the virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC). When APIC virtualization is enabled, the processor emulates many accesses to the APIC, tracks the state of the virtual APIC, and delivers virtual interrupts — all in VMX non-root operation without a VM exit.

Details of the APIC virtualization are given in Chapter 32.

28.5.6 VM Functions

A **VM function** is an operation provided by the processor that can be invoked from VMX non-root operation without a VM exit. VM functions are enabled and configured by the settings of different fields in the VMCS. Software in VMX non-root operation invokes a VM function with the **VMFUNC** instruction; the value of EAX selects the specific VM function being invoked.

Section 28.5.6.1 explains how VM functions are enabled. Section 28.5.6.2 specifies the behavior of the VMFUNC instruction. Section 28.5.6.3 describes a specific VM function called **EPTP switching**.

28.5.6.1 Enabling VM Functions

Software enables VM functions generally by setting the “enable VM functions” VM-execution control. A specific VM function is enabled by setting the corresponding VM-function control.

Suppose, for example, that software wants to enable EPTP switching (VM function 0; see Section 27.6.14). To do so, it must set the “activate secondary controls” VM-execution control (bit 31 of the primary processor-based VM-execution controls), the “enable VM functions” VM-execution control (bit 13 of the secondary processor-based VM-execution controls) and the “EPTP switching” VM-function control (bit 0 of the VM-function controls).

28.5.6.2 General Operation of the VMFUNC Instruction

The VMFUNC instruction causes an invalid-opcode exception (#UD) if the “enable VM functions” VM-execution controls is 0¹ or the value of EAX is greater than 63 (only VM functions 0–63 can be enable). Otherwise, the instruction causes a VM exit if the bit at position EAX is 0 in the VM-function controls (the selected VM function is not enabled). If such a VM exit occurs, the basic exit reason used is 59 (3BH), indicating “VMFUNC”, and the length of the VMFUNC instruction is saved into the VM-exit instruction-length field. If the instruction causes neither an invalid-opcode exception nor a VM exit due to a disabled VM function, it performs the functionality of the VM function specified by the value in EAX.

Individual VM functions may perform additional fault checking (e.g., one might cause a general-protection exception if CPL > 0). In addition, specific VM functions may include checks that might result in a VM exit. If such a VM exit occurs, VM-exit information is saved as described in the previous paragraph. The specification of a VM function may indicate that additional VM-exit information is provided.

The specific behavior of the EPTP-switching VM function (including checks that result in VM exits) is given in Section 28.5.6.3.

28.5.6.3 EPTP Switching

EPTP switching is VM function 0. This VM function allows software in VMX non-root operation to load a new value for the EPT pointer (EPTP), thereby establishing a different EPT paging-structure hierarchy (see Section 31.3 for details of the operation of EPT). Software is limited to selecting from a list of potential EPTP values configured in advance by software in VMX root operation.

Specifically, the value of ECX is used to select an entry from the EPTP list, the 4-KByte structure referenced by the EPTP-list address (see Section 27.6.14; because this structure contains 512 8-Byte entries, VMFUNC causes a VM exit if ECX ≥ 512). The EPTP value in the selected entry is evaluated to determine whether it is valid for EPTP switching: a value is valid if (1) it is the same as the current EPTP value in bits 5:3 (these bits specify the EPT page-walk length); and (2) it would not cause VM entry to fail (see Section 29.2.1.1). If the value is invalid, a VM exit occurs. Otherwise, the value is stored in the EPTP field of the current VMCS and is used for subsequent accesses using guest-physical addresses. The following pseudocode provides details:

IF ECX ≥ 512

-
1. “Enable VM functions” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “enable VM functions” VM-execution control were 0. See Section 27.6.2.

```

THEN VM exit;
ELSE
    tent_EPTP := 8 bytes from EPTP-list address + 8 * ECX;
    IF tent_EPTP is not a valid EPTP value (would cause VM entry to fail if in EPTP or change EPT page-walk length)
    THEN VM exit;
    ELSE
        write tent_EPTP to the EPTP field in the current VMCS;
        use tent_EPTP as the new EPTP value for address translation;
        IF processor supports the 1-setting of the "EPT-violation #VE" VM-execution control
        THEN
            write ECX[15:0] to EPTP-index field in current VMCS;
            use ECX[15:0] as EPTP index for subsequent EPT-violation virtualization exceptions (see Section 28.5.7.2);
        FI;
    FI;
FI;

```

Execution of the EPTP-switching VM function does not modify the state of any registers; no flags are modified.

If the "Intel PT uses guest physical addresses" VM-execution control is 1 and IA32_RTIT_CTL.TraceEn = 1, any execution of the EPTP-switching VM function causes a VM exit.¹

As noted in Section 28.5.6.2, an execution of the EPTP-switching VM function that causes a VM exit (as specified above), uses the basic exit reason 59, indicating "VMFUNC". The length of the VMFUNC instruction is saved into the VM-exit instruction-length field. No additional VM-exit information is provided.

An execution of VMFUNC loads EPTP from the EPTP list (and thus does not cause a fault or VM exit) is called an **EPTP-switching VMFUNC**. After an EPTP-switching VMFUNC, control passes to the next instruction. The logical processor starts creating and using guest-physical and combined mappings associated with the new value of bits 51:12 of EPTP; the combined mappings created and used are associated with the current VPID and PCID (these are not changed by VMFUNC).² If the "enable VPID" VM-execution control is 0, an EPTP-switching VMFUNC invalidates combined mappings associated with VPID 0000H (for all PCIDs and for all EPTRTA values, where EPTRTA is the value of bits 51:12 of EPTP).

Because an EPTP-switching VMFUNC may change the translation of guest-physical addresses, it may affect use of the guest-physical address in CR3. The EPTP-switching VMFUNC cannot itself cause a VM exit due to an EPT violation or an EPT misconfiguration due to the translation of that guest-physical address through the new EPT paging structures. The following items provide details that apply if CR0.PG = 1:

- If 32-bit paging or 4-level paging³ is in use (either CR4.PAE = 0 or IA32_EFER.LMA = 1), the next memory access with a linear address uses the translation of the guest-physical address in CR3 through the new EPT paging structures. As a result, this access may cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during that translation.
- If PAE paging is in use (CR4.PAE = 1 and IA32_EFER.LMA = 0), an EPTP-switching VMFUNC **does not** load the four page-directory-pointer-table entries (PDPTEs) from the guest-physical address in CR3. The logical processor continues to use the four guest-physical addresses already present in the PDPTEs. The guest-physical address in CR3 is not translated through the new EPT paging structures (until some operation that would load the PDPTEs).

The EPTP-switching VMFUNC cannot itself cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during the translation of a guest-physical address in any of the PDPTEs. A subsequent memory access with a linear address uses the translation of the guest-physical address in the appropriate PDPTE through the new EPT paging structures. As a result, such an access may cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during that translation.

If an EPTP-switching VMFUNC establishes an EPTP value that enables accessed and dirty flags for EPT (by setting bit 6), subsequent memory accesses may fail to set those flags as specified if there has been no appropriate execu-

-
1. Such a VM exit ensures the proper recording of trace data that might otherwise be lost during the change of EPT paging-structure hierarchy. Software handling the VM exit can change emulate the VM function and then resume the guest.
 2. If the "enable VPID" VM-execution control is 0, the current VPID is 0000H; if CR4.PCIDE = 0, the current PCID is 000H.
 3. Earlier versions of this manual used the term "IA-32e paging" to identify 4-level paging.

tion of INVEPT since the last use of an EPTP value that does not enable accessed and dirty flags for EPT (because bit 6 is clear) and that is identical to the new value on bits 51:12.

If the processor supports the 1-setting of the “EPT-violation #VE” VM-execution control, an EPTP-switching VMFUNC loads the value in ECX[15:0] into the EPTP-index field in current VMCS. Subsequent EPT-violation virtualization exceptions will save this value into the virtualization-exception information area (see Section 28.5.7.2).

28.5.7 Virtualization Exceptions

A **virtualization exception** is a new processor exception. It uses vector 20 and is abbreviated #VE.

A virtualization exception can occur only in VMX non-root operation. Virtualization exceptions occur only with certain settings of certain VM-execution controls. Generally, these settings imply that certain conditions that would normally cause VM exits instead cause virtualization exceptions

In particular, the 1-setting of the “EPT-violation #VE” VM-execution control causes some EPT violations to generate virtualization exceptions instead of VM exits. Section 28.5.7.1 provides the details of how the processor determines whether an EPT violation causes a virtualization exception or a VM exit.

When the processor encounters a virtualization exception, it saves information about the exception to the virtualization-exception information area; see Section 28.5.7.2.

After saving virtualization-exception information, the processor delivers a virtualization exception as it would any other exception; see Section 28.5.7.3 for details.

28.5.7.1 Convertible EPT Violations

If the “EPT-violation #VE” VM-execution control is 0 (e.g., on processors that do not support this feature), EPT violations always cause VM exits. If instead the control is 1, certain EPT violations may be converted to cause virtualization exceptions instead; such EPT violations are **convertible**.

The values of certain EPT paging-structure entries determine which EPT violations are convertible. Specifically, bit 63 of certain EPT paging-structure entries may be defined to mean **suppress #VE**:

- If bits 2:0 of an EPT paging-structure entry are all 0, the entry is not **present**.¹ If the processor encounters such an entry while translating a guest-physical address, it causes an EPT violation. The EPT violation is convertible if and only if bit 63 of the entry is 0.
- If an EPT paging-structure entry is present, the following cases apply:
 - If the value of the EPT paging-structure entry is not supported, the entry is **misconfigured**. If the processor encounters such an entry while translating a guest-physical address, it causes an EPT misconfiguration (not an EPT violation). EPT misconfigurations always cause VM exits.
 - If the value of the EPT paging-structure entry is supported, the following cases apply:
 - If bit 7 of the entry is 1, or if the entry is an EPT PTE, the entry maps a page. If the processor uses such an entry to translate a guest-physical address, and if an access to that address causes an EPT violation, the EPT violation is convertible if and only if bit 63 of the entry is 0.
 - If bit 7 of the entry is 0 and the entry is not an EPT PTE, the entry references another EPT paging structure. The processor does not use the value of bit 63 of the entry to determine whether any subsequent EPT violation is convertible.

If an access to a guest-physical address causes an EPT violation, bit 63 of exactly one of the EPT paging-structure entries used to translate that address is used to determine whether the EPT violation is convertible: either a entry that is not present (if the guest-physical address does not translate to a physical address) or an entry that maps a page (if it does).

A convertible EPT violation instead causes a virtualization exception if the following all hold:

- CR0.PE = 1;

1. If the “mode-based execute control for EPT” VM-execution control is 1, an EPT paging-structure entry is present if any of bits 2:0 or bit 10 is 1.

- the logical processor is not in the process of delivering an event;
- the EPT violation did not cause a shadow stack to become prematurely busy (see Section 28.4.3);
- the EPT violation does not result from the output process of Intel Processor Trace (Section 28.5.4); and
- the 32 bits at offset 4 in the virtualization-exception information area are all 0.

Delivery of virtualization exceptions writes the value FFFFFFFFH to offset 4 in the virtualization-exception information area (see Section 28.5.7.2). Thus, once a virtualization exception occurs, another can occur only if software clears this field.

28.5.7.2 Virtualization-Exception Information

Virtualization exceptions save data into the virtualization-exception information area (see Section 27.6.19). Table 28-23 enumerates the data saved and the format of the area.

Table 28-23. Format of the Virtualization-Exception Information Area

Byte Offset	Contents
0	The 32-bit value that would have been saved into the VMCS as an exit reason had a VM exit occurred instead of the virtualization exception. For EPT violations, this value is 48 (00000030H)
4	FFFFFFFFH
8	The 64-bit value that would have been saved into the VMCS as an exit qualification had a VM exit occurred instead of the virtualization exception
16	The 64-bit value that would have been saved into the VMCS as a guest-linear address had a VM exit occurred instead of the virtualization exception
24	The 64-bit value that would have been saved into the VMCS as a guest-physical address had a VM exit occurred instead of the virtualization exception
32	The current 16-bit value of the EPTP index VM-execution control (see Section 27.6.19 and Section 28.5.6.3)

A VMM may allow guest software to access the virtualization-exception information area. If it does, the guest software may modify that memory (e.g., to clear the 32-bit value at offset 4; see Section 28.5.7.1). (This is an exception to the general requirement given in Section 27.11.4.)

28.5.7.3 Delivery of Virtualization Exceptions

After saving virtualization-exception information, the processor treats a virtualization exception as it does other exceptions:

- If bit 20 (#VE) is 1 in the exception bitmap in the VMCS, a virtualization exception causes a VM exit (see below). If the bit is 0, the virtualization exception is delivered using [vector 20](#).
- Virtualization exceptions produce no error code. Delivery of a virtualization exception pushes no error code on the stack.
- With respect to double faults, virtualization exceptions have the same severity as page faults. If delivery of a virtualization exception encounters a nested fault that is either contributory or a page fault, a double fault (#DF) is generated. See Chapter 7, “Interrupt 8—Double Fault Exception (#DF)” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

It is not possible for a virtualization exception to be encountered while delivering another exception (see Section 28.5.7.1).

If a virtualization exception causes a VM exit directly (because bit 20 is 1 in the exception bitmap), information about the exception is saved normally in the [exiting-event identification](#) field in the VMCS (see Section 30.2.2). Specifically, the event is reported as a hardware exception with vector 20 and no error code. Bit 12 of the field (NMI unblocking due to IRET) is set normally.

If a virtualization exception causes a VM exit indirectly (because bit 20 is 0 in the exception bitmap and delivery of the exception generates an event that causes a VM exit), information about the exception is saved normally in the **original-event identification** field in the VMCS (see Section 30.2.4). Specifically, the event is reported as a hardware exception with vector 20 and no error code.

28.5.8 PASID Translation

The ENQCMD and ENQCMDS instructions each performs a 64-byte enqueue store that includes a 20-bit PASID value in bits 19:0. For ENQCMD, the PASID is normally the value of IA32_PASID[19:0], while for ENQCMDS, the PASID is normally read from memory.

If the “PASID translation” VM-execution control is 1, the PASID value identified in the previous paragraph is treated as a **guest PASID**. PASID translation converts this guest PASID to a 20-bit **host PASID**. After this translation, the enqueue store is performed, using the host PASID in place of the guest PASID.

PASID translation is implemented by two hierarchies of data structures (**PASID-translation hierarchies**) configured by a VMM. Guest PASIDs 00000H to 7FFFFH are translated through the low PASID-translation hierarchy, while guest PASIDs 80000 to FFFFFH are translated through the high PASID-translation hierarchy.

The root of each PASID-translation hierarchy is a 4-KByte **PASID directory**. The low PASID directory is located at the low PASID directory address, and the high PASID directory is located at the high PASID directory address (these physical addresses are VM-execution control fields in the VMCS). A PASID directory comprises 512 8-byte entries, each of which has the following format:

- Bit 0 is the entry’s present bit. The entry is used only if this bit is 1.
- Bits 11:1 are reserved and must be 0.
- Bits MAXPHYADDR-1:12 specify the 4-KByte aligned address of a PASID table (see below), where MAXPHYADDR is defined as follows:
 - Usually, MAXPHYADDR is the processor’s supported physical-address width as enumerated in CPUID.80000008H:EAX[7:0] (this width is at most 52).
 - If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.¹
- Bits 63:MAXPHYADDR are reserved and must be 0.

A PASID-translation hierarchy also includes up to 512 4-KByte **PASID tables**; each of these is referenced by a PASID directory entry (see above). A PASID table comprises 1024 4-byte entries, each of which has the following format:

- Bits 19:0 are the host PASID specified by the entry.
- Bits 30:20 are reserved and must be 0.
- Bit 31 is the entry’s valid bit. The entry is used only if this bit is 1.

When PASID translation is enabled, the guest PASID determined by the instruction (see above) is converted to a host PASID using the following process:

- If bit 19 of guest PASID is clear, the low PASID directory is used; otherwise, the high PASID directory is used.
- Bits 18:10 of the guest PASID select an entry from the PASID directory. A VM exit occurs if the entry’s present bit is clear or if any reserved bit is set. Otherwise, bits MAXPHYADDR-1:0 of the entry (with bit 0 cleared) contain the physical address of a PASID table.
- Bits 9:0 of the guest PASID select an entry from the PASID table. A VM exit occurs if the entry’s valid bit is clear or if any reserved bit is set. Otherwise, bits 19:0 of the entry are the host PASID.

If PASID translation results in a VM exit (due to a present or valid bit being clear, or a reserved bit being set), the instruction does not complete and no enqueue store is performed.

1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

28.6 UNRESTRICTED GUESTS

The first processors to support VMX operation require CR0.PE and CR0.PG to be 1 in VMX operation (see Section 26.8). This restriction implies that guest software cannot be run in unpaged protected mode or in real-address mode. Later processors support a VM-execution control called “unrestricted guest”.¹ If this control is 1, CR0.PE and CR0.PG may be 0 in VMX non-root operation. Such processors allow guest software to run in unpaged protected mode or in real-address mode. The following items describe the behavior of such software:

- The MOV CR0 instructions does not cause a general-protection exception simply because it would set either CR0.PE and CR0.PG to 0. See Section 28.3 for details.
- A logical processor treats the values of CR0.PE and CR0.PG in VMX non-root operation just as it does outside VMX operation. Thus, if CR0.PE = 0, the processor operates as it does normally in real-address mode (for example, it uses the 16-bit **interrupt table** to deliver interrupts and exceptions). If CR0.PG = 0, the processor operates as it does normally when paging is disabled.
- Processor operation is modified by the fact that the processor is in VMX non-root operation and by the settings of the VM-execution controls just as it is in protected mode or when paging is enabled. Instructions, interrupts, and exceptions that cause VM exits in protected mode or when paging is enabled also do so in real-address mode or when paging is disabled. The following examples should be noted:
 - If CR0.PG = 0, page faults do not occur and thus cannot cause VM exits.
 - If CR0.PE = 0, invalid-TSS exceptions do not occur and thus cannot cause VM exits.
 - If CR0.PE = 0, the following instructions cause invalid-opcode exceptions and do not cause VM exits: INVEPT, INVVPID, LLDT, LTR, SLDT, STR, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON.
- If CR0.PG = 0, each linear address is passed directly to the EPT mechanism for translation to a physical address.² The guest memory type passed on to the EPT mechanism is WB (writeback).

1. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “unrestricted guest” VM-execution control were 0. See Section 27.6.2.

2. As noted in Section 29.2.1.1, the “enable EPT” VM-execution control must be 1 if the “unrestricted guest” VM-execution control is 1.

23. Updates to Chapter 29, Volume 3C

Change bars and text show changes to Chapter 29 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- Updated Section 29.2.1.3, "VM-Entry Control Fields" for FRED.
- Added "load FRED" to Section 29.2.2, "Checks on Host Control Registers, MSRs, and SSP".
- Added that bit 32 of the CR4 field is 0 to the list of conditions required if "host address-space size" VM-exit control is 0 in Section 29.2.4, "Checks Related to Address-Space Size".
- Added a check performed using bit 32 in the CR4 field corresponding to CR4.FRED in Section 29.3.1.1, "Checks on Guest Control Registers, Debug Registers, and MSRs" and added "load FRED".
- Added content for FRED transitions in Section 29.3.1.2, "Checks on Guest Segment Registers".
- Replaced IDT-vectoring with original-event, VM-entry with injected-event, and VM-exit with exiting-event in
 - Section 29.3.1.4, "Checks on Guest RIP, RFLAGS, and SSP"
 - Section 29.3.1.5, "Checks on Guest Non-Register State"
 - Section 29.5, "Trace-Address Pre-Translation (TAPT)"
 - Section 29.6.1.2, "VM Exits During Event Injection"
 - Section 29.6.1.3, "Event Injection for VM Entries to Real-Address Mode"
 - Section 29.6.2, "Injection of Pending MTF VM Exits"
 - Section 29.8, "VM-Entry Failures During or After Loading Guest State"
- Added "load FRED" in Section 29.3.2.1, "Loading Guest Control Registers, Debug Registers, and MSRs".
- Added event type 7 in Section 29.6, "Event Injection".
- Added content for FRED transitions in Section 29.6.1, "Injection of Interrupts and Exceptions".
- Documented FRED event delivery in Section 29.6.1.1, "Details of Event Injection".
- Updated Section 29.7, "Special Features of VM Entry" and its subsections replacing references to vectoring with injecting for FRED.
- In Section 29.9, "Machine-Check Events During VM Entry", removed references to IDT.

Software can enter VMX non-root operation using either of the VM-entry instructions VMLAUNCH and VMRESUME. VMLAUNCH can be used only with a VMCS whose launch state is clear and VMRESUME can be used only with a VMCS whose the launch state is launched. VMLAUNCH should be used for the first VM entry after VMCLEAR; VMRESUME should be used for subsequent VM entries with the same VMCS.

Each VM entry performs the following steps in the order indicated:

1. Basic checks are performed to ensure that VM entry can commence (Section 29.1).
2. The control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation and that the VMCS is correctly configured to support the next VM exit (Section 29.2).
3. The following are performed in parallel or in any order (Section 29.3):
 - The guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures.
 - Processor state is loaded from the guest-state area and based on controls in the VMCS.
 - Address-range monitoring is cleared.
4. MSRs are loaded from the VM-entry MSR-load area (Section 29.4).
5. If VMLAUNCH is being executed, the launch state of the VMCS is set to “launched.”
6. If the “Intel PT uses guest physical addresses” VM-execution control is 1, trace-address pre-translation (TAPT) may occur (see Section 28.5.4 and Section 29.5).
7. An event may be injected in the guest context (Section 29.6).

Steps 1–4 above perform checks that may cause VM entry to fail. Such failures occur in one of the following three ways:

- Some of the checks in Section 29.1 may generate ordinary faults (for example, an invalid-opcode exception). Such faults are delivered normally.
- Some of the checks in Section 29.1 and all the checks in Section 29.2 cause control to pass to the instruction following the VM-entry instruction. The failure is indicated by setting RFLAGS.ZF¹ (if there is a current VMCS) or RFLAGS.CF (if there is no current VMCS). If there is a current VMCS, an error number indicating the cause of the failure is stored in the VM-instruction error field. See Chapter 32 for the error numbers.
- The checks in Section 29.3 and Section 29.4 cause processor state to be loaded from the host-state area of the VMCS (as would be done on a VM exit). Information about the failure is stored in the VM-exit information fields. See Section 29.8 for details.

EFLAGS.TF = 1 causes a VM-entry instruction to generate a single-step debug exception only if failure of one of the checks in Section 29.1 and Section 29.2 causes control to pass to the following instruction. A VM-entry does not generate a single-step debug exception in any of the following cases: (1) the instruction generates a fault; (2) failure of one of the checks in Section 29.3 or in loading MSRs causes processor state to be loaded from the host-state area of the VMCS; or (3) the instruction passes all checks in Section 29.1, Section 29.2, and Section 29.3 and there is no failure in loading MSRs.

Section 34.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, code running in SMM returns using VM entries instead of the RSM instruction. A VM entry **returns from SMM** if it is executed in SMM and the “entry to SMM” VM-entry control is 0. VM entries that return from SMM differ from ordinary VM entries in ways that are detailed in Section 34.15.4.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

29.1 BASIC VM-ENTRY CHECKS

Before a VM entry commences, the current state of the logical processor is checked in the following order:

1. If the logical processor is in virtual-8086 mode or compatibility mode, an invalid-opcode exception is generated.
2. If the current privilege level (CPL) is not zero, a general-protection exception is generated.
3. If there is no current VMCS, RFLAGS.CF is set to 1 and control passes to the next instruction.
4. If there is a current VMCS but the current VMCS is a shadow VMCS (see Section 27.10), RFLAGS.CF is set to 1 and control passes to the next instruction.
5. If there is a current VMCS that is not a shadow VMCS, the following conditions are evaluated in order; any of these cause VM entry to fail:
 - a. If there is MOV-SS blocking (see Table 27-3).
 - b. If the VM entry is invoked by VMLAUNCH and the VMCS launch state is not clear.
 - c. If the VM entry is invoked by VMRESUME and the VMCS launch state is not launched.

If any of these checks fail, RFLAGS.ZF is set to 1 and control passes to the next instruction. An error number indicating the cause of the failure is stored in the VM-instruction error field. See Chapter 32 for the error numbers.

29.2 CHECKS ON VMX CONTROLS AND HOST-STATE AREA

If the checks in Section 29.1 do not cause VM entry to fail, the control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation, that the VMCS is correctly configured to support the next VM exit, and that, after the next VM exit, the processor's state is consistent with the Intel 64 and IA-32 architectures.

VM entry fails if any of these checks fail. When such failures occur, control is passed to the next instruction, RFLAGS.ZF is set to 1 to indicate the failure, and the VM-instruction error field is loaded with an error number that indicates whether the failure was due to the controls or the host-state area (see Chapter 32).

These checks may be performed in any order. Thus, an indication by error number of one cause (for example, host state) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same VMCS. Some checks prevent establishment of settings (or combinations of settings) that are currently reserved. Future processors may allow such settings (or combinations) and may not perform the corresponding checks. The correctness of software should not rely on VM-entry failures resulting from the checks documented in this section.

The checks on the controls and the host-state area are presented in Section 29.2.1 through Section 29.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

29.2.1 Checks on VMX Controls

This section identifies VM-entry checks on the VMX control fields.

Some of these fields contain physical addresses. These fields are checked against the allowed address width:

- If IA32_VMX_BASIC[48] is read as 1 (implying that the processor does not support Intel 64 architecture; see Section A.1, "Basic VMX Information"), these addresses must not set any bits in the range 63:32.
- Otherwise, the widths of physical addresses are limited by MAXPHYADDR, a value derived from the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.¹ Physical address should not set bits in the range 63:MAXPHYADDR.

29.2.1.1 VM-Execution Control Fields

VM entries perform the following checks on the VM-execution control fields:¹

- Reserved bits in the pin-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.1).
- Reserved bits in the primary processor-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.2).
- If the “activate secondary controls” primary processor-based VM-execution control is 1, reserved bits in the secondary processor-based VM-execution controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.3.3).

If the “activate secondary controls” primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary processor-based VM-execution controls. The logical processor operates as if all the secondary processor-based VM-execution controls were 0.

- If the “activate tertiary controls” primary processor-based VM-execution control is 1, reserved bits in the tertiary processor-based VM-execution controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.3.4).

If the “activate tertiary controls” primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the tertiary processor-based VM-execution controls. The logical processor operates as if all the tertiary processor-based VM-execution controls were 0.

- The CR3-target count must not be greater than 4. Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC to determine the number of values supported (see Appendix A.6).
- Bits 11:0 of the following fields must be 0, and the fields are also subject to the checks on physical-address width described above in Section 28.2.1:
 - If the “use I/O bitmaps” VM-execution control is 1, each I/O-bitmap address.
 - If the “use MSR bitmaps” VM-execution control is 1, the MSR-bitmap address.
 - If the “use TPR shadow” VM-execution control is 1, the virtual-APIC address.
If the checks on that address are satisfied, bytes 3:1 of VTPR (see Section 31.1.1) may be cleared (behavior may be implementation-specific).
The clearing of these bytes may occur even if the VM entry fails. This is true either if the failure causes control to pass to the instruction following the VM-entry instruction or if it causes processor state to be loaded from the host-state area of the VMCS.
 - If the “virtualize APIC-accesses” VM-execution control is 1, the APIC-access address.
 - If the “enable PML” VM-execution control is 1, the PML address.
 - If the “sub-page write permissions for EPT” VM-execution control is 1, the SPPTP VM-execution control field (see Table 26-11 in Section 26.6.21).
 - If the “enable VM functions” processor-based VM-execution control is 1 and the “EPTP switching” VM-function control is 1, the EPTP-list address.
 - If the “VMCS shadowing” VM-execution control is 1, the VMREAD-bitmap and VMWRITE-bitmap addresses.
 - If the “EPT-violation #VE” VM-execution control is 1, the virtualization-exception information address.
 - If the “PASID translation” VM-execution control is 1, the low PASID directory address and the high PASID directory address.

1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

1. If the “activate secondary controls” primary processor-based VM-execution control is 0, VM entry operates as if each secondary processor-based VM-execution control were 0. Similarly, if the “activate tertiary controls” primary processor-based VM-execution control is 0, VM entry operates as if each tertiary processor-based VM-execution control were 0. See Section 27.6.2.

- If the “use TPR shadow” VM-execution control is 1 and the “virtual-interrupt delivery” VM-execution control is 0, bits 31:4 of the TPR threshold VM-execution control field must be 0.
- The following check is performed if the “use TPR shadow” VM-execution control is 1 and the “virtualize APIC accesses” and “virtual-interrupt delivery” VM-execution controls are both 0: the value of bits 3:0 of the TPR threshold VM-execution control field should not be greater than the value of bits 7:4 of VTPR (see Section 32.1.1).
- If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” VM-execution control must be 0.
- If the “virtual NMIs” VM-execution control is 0, the “NMI-window exiting” VM-execution control must be 0.
- If the “use TPR shadow” VM-execution control is 0, the following VM-execution controls must also be 0: “virtualize x2APIC mode”, “APIC-register virtualization”, “virtual-interrupt delivery”, and “IPI virtualization”.
- If the “use TPR shadow” VM-execution control is 0, the following VM-execution controls must also be 0: “virtualize x2APIC mode”, “APIC-register virtualization”, “virtual-interrupt delivery”, and “IPI virtualization”.
- If the “virtualize x2APIC mode” VM-execution control is 1, the “virtualize APIC accesses” VM-execution control must be 0.
- If the “virtual-interrupt delivery” VM-execution control is 1, the “external-interrupt exiting” VM-execution control must be 1.
- If the “process posted interrupts” VM-execution control is 1, the following must be true:
 - The “virtual-interrupt delivery” VM-execution control is 1.
 - The “acknowledge interrupt on exit” VM-exit control is 1.
 - The posted-interrupt notification vector has a value in the range 0–255 (bits 15:8 are all 0).
 - Bits 5:0 of the posted-interrupt descriptor address are all 0.
 - The posted-interrupt descriptor address satisfies the checks on physical-address width described above in Section 29.2.1.
- If the “IPI virtualization” VM-execution control is 1, the following must be true:
 - Bits 2:0 of the PID-pointer table address are all 0.
 - The PID-pointer table address satisfies the checks on physical-address width described above in Section 29.2.1.
 - The address of the last entry in the PID-pointer table satisfies the checks on physical-address width described above. (This address is the PID-pointer table address plus 8 times the last PID-pointer index.)
- If the “enable VPID” VM-execution control is 1, the value of the VPID VM-execution control field must not be 0000H.
- If the “enable EPT” VM-execution control is 1, the EPTP VM-execution control field (see Table 27-9 in Section 27.6.11) must satisfy the following checks:
 - The EPT memory type (bits 2:0) must be a value supported by the processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10).
 - Bits 5:3 must contain a value 1 less than an EPT page-walk length supported by the processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Section 31.3.2 and Appendix A.10).
 - Bit 6 (enable bit for accessed and dirty flags for EPT) must be 0 if bit 21 of the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10) is read as 0, indicating that the processor does not support accessed and dirty flags for EPT.
 - Reserved bits 11:7 must all be 0.
 - The 4-KByte-aligned address defined by bits 63:12 must satisfy the checks on physical-address width described above in Section 29.2.1.
- The “enable EPT” VM-execution control must be 1 if any of the following VM-execution controls is 1: “enable PML”, “unrestricted guest”, “mode-based execute control for EPT”, “sub-page write permissions for EPT”, “Intel PT uses guest physical addresses”, “enable HLAT”, “EPT paging-write control”, or “guest-paging verification.”
- The following checks apply if the logical processor is in SEAM at the time of VM entry:
 - The “enable EPT” VM-execution control must be 1.

- Bits 11:0 and bits 63:M must be zero in the SEAM shared EPT pointer, where M is the value enumerated by CPUID.80000008H:EAX[7:0].
- If the “enable VM functions” processor-based VM-execution control is 1, reserved bits in the VM-function controls must be clear. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.11). In addition, the following check is performed based on the setting of bits in the VM-function controls (see Section 27.6.14):
 - If “EPTP switching” VM-function control is 1, the “enable EPT” VM-execution control must also be 1.
If the “enable VM functions” processor-based VM-execution control is 0, no checks are performed on the VM-function controls.
- If the logical processor is operating with Intel PT enabled (if IA32_RTIT_CTL.TraceEn = 1) at the time of VM entry, the “load IA32_RTIT_CTL” VM-entry control must be 0.
- If the “Intel PT uses guest physical addresses” VM-execution control is 1, the “load IA32_RTIT_CTL” VM-entry control and the “clear IA32_RTIT_CTL” VM-exit control must both be 1.
- If the “use TSC scaling” VM-execution control is 1, the TSC-multiplier must not be zero.
- If the “enable HLAT” VM-execution control is 1, the following bits in the HLATP VM-execution control field (see Table 27-12 in Section 27.6.22) must be zero: bits 2:0, bits 11:5. In addition, the 4-KByte-aligned address defined by bits 63:12 must satisfy the checks on physical-address width described above in Section 29.2.1.

29.2.1.2 VM-Exit Control Fields

VM entries perform the following checks on the VM-exit control fields.

- Reserved bits in the primary VM-exit controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.4.1).
- If the “activate secondary controls” primary VM-exit control is 1, reserved bits in the secondary VM-exit controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.4.2).
- If the “activate secondary controls” primary VM-exit control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary VM-exit controls. The logical processor operates as if all the secondary VM-exit controls were 0.
- If the “activate VMX-preemption timer” VM-execution control is 0, the “save VMX-preemption timer value” VM-exit control must also be 0.
- The following checks are performed for the VM-exit MSR-store address if the VM-exit MSR-store count field is non-zero:
 - The lower 4 bits of the VM-exit MSR-store address must be 0. The address must satisfy the checks on physical-address width described above in Section 29.2.1.
 - The address of the last byte in the VM-exit MSR-store area must satisfy the checks on physical-address width described above. The address of this last byte is VM-exit MSR-store address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor’s physical-address width.)
- The following checks are performed for the VM-exit MSR-load address if the VM-exit MSR-load count field is non-zero:
 - The lower 4 bits of the VM-exit MSR-load address must be 0. The address must satisfy the checks on physical-address width described above.
 - The address of the last byte in the VM-exit MSR-load area must satisfy the checks on physical-address width described above. The address of this last byte is VM-exit MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor’s physical-address width.)

29.2.1.3 VM-Entry Control Fields

VM entries perform the following checks on the VM-entry control fields.

- Reserved bits in the VM-entry controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.5).

- Fields relevant to VM-entry event injection must be set properly. These fields are the [injected-event identification](#) field (see Table 27-18 in Section 27.8.3, “VM-Entry Controls for Event Injection”), the [injected-event error code](#), and the VM-entry instruction length. If the valid bit (bit 31) in the [injected-event identification](#) field is 1, the following must hold:
 - The field’s [event](#) type (bits 10:8) is not set to a reserved value. Value 1 is reserved on all logical processors; value 7 (other event) is reserved on logical processors that support [neither](#) the 1-setting of the “monitor trap flag” VM-execution control [nor FRED transitions](#).
 - The field’s vector (bits 7:0) is consistent with the [event](#) type:
 - If the [event](#) type is non-maskable interrupt (NMI), the vector is 2.
 - If the [event](#) type is hardware exception, the vector is at most 31.
 - If the [event](#) type is other event, the vector is 0 ([indicating a pending MTF VM exit, only if “monitor trap flag” is supported](#)) or, if [FRED transitions would be enabled following VM entry \(bit 32 of the CR4 field — FRED — in the guest-state area is 1\)](#), the vector is 1 ([indicating SYSCALL](#)) or 2 ([indicating SYSENTER](#)).
 - The field’s deliver-error-code bit (bit 11) is 1 if each of the following holds: (1) the [event](#) type is hardware exception; (2) either (a) the “unrestricted guest” VM-execution control is 0; or (b) bit 0 (corresponding to CR0.PE) is set in the CR0 field in the guest-state area; (3) IA32_VMX_BASIC[56] is read as 0 (see Appendix A.1); and (4) the vector indicates one of the following exceptions: #DF (vector 8), #TS (10), #NP (11), #SS (12), #GP (13), #PF (14), or #AC (17).
 - The field’s deliver-error-code bit is 0 if any of the following holds: (1) the [event](#) type is not hardware exception; (2) both (a) the “unrestricted guest” VM-execution control is 1; and (b) bit 0 is clear in the CR0 field in the guest-state area; or (3) IA32_VMX_BASIC[56] is read as 0 and the vector is in one of the following ranges: 0–7, 9, 15, 16, or 18–31.

NOTE

IA32_VMX_BASIC[56] will be read as 1 on any processor that supports FRED transitions.

- The field’s nested-exception bit (bit 13) is 0 if either of the following hold: (1) the event type is not hardware exception; or (2) IA32_VMX_BASIC[58] is read as 0.
- Reserved bits in the field (30:14 and 12) are 0.
- If the deliver-error-code bit (bit 11) is 1, bits 31:16 of the [injected-event](#) error-code field are 0.
- The VM-entry instruction-length field is in the range 0–15 if the event type is software interrupt, software exception, or privileged software exception; or if the event type is other event and the vector is either 1 ([SYSCALL](#)) or 2 ([SYSENTER](#)). A VM-entry instruction length of 0 is allowed only if IA32_VMX_MISC[30] is read as 1; see Appendix A.6.
- The following checks are performed for the VM-entry MSR-load address if the VM-entry MSR-load count field is non-zero:
 - The lower 4 bits of the VM-entry MSR-load address must be 0. The address must satisfy the checks on physical-address width described above in Section 29.2.1.
 - The address of the last byte in the VM-entry MSR-load area must satisfy the checks on physical-address width described above. The address of this last byte is VM-entry MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor’s physical-address width.)
 If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1, “Basic VMX Information”.
- If the processor is not in SMM, the “entry to SMM” and “deactivate dual-monitor treatment” VM-entry controls must be 0.
- The “entry to SMM” and “deactivate dual-monitor treatment” VM-entry controls cannot both be 1.

29.2.2 Checks on Host Control Registers, MSRs, and SSP

The following checks are performed on fields in the host-state area that correspond to control registers and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 26.8).¹

- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 26.8).
- If bit 23 in the CR4 field (corresponding to CET) is 1, bit 16 in the CR0 field (WP) must also be 1.
- On processors that support Intel 64 architecture, the CR3 field must be such that bits reserved in CR3 are 0.¹ In particular, bits 63:32 are checked for allowed address width:
 - If IA32_VMX_BASIC[48] is read as 1 (implying that the processor does not support Intel 64 architecture; see Appendix A.1, “Basic VMX Information”), the field must not set any of bits 63:32.
 - Otherwise, the field’s width is limited by MAXPHYADDR, which is derived from the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM. The field should not set bits in the range 63:MAXPHYADDR.
- On processors that support Intel 64 architecture, the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
- If the “load IA32_PERF_GLOBAL_CTRL” VM-exit control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 22-3).
- If the “load IA32_PAT” VM-exit control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the “load IA32_EFER” VM-exit control is 1, bits reserved in the IA32_EFER MSR must be 0 in the field for that register. In addition, the values of the LMA and LME bits in the field must each be that of the “host address-space size” VM-exit control.
- If the “load CET state” VM-exit control is 1, the IA32_S_CET field must not set any bits reserved in the IA32_S_CET MSR, and bit 10 (corresponding to SUPPRESS) and bit 11 (TRACKER) in the field cannot both be set.
- If the “load CET state” VM-exit control is 1, bits 1:0 must be 0 in the SSP field.
- If the “load PKRS” VM-exit control is 1, bits 63:32 must be 0 in the IA32_PKRS field.
- If the “load FRED” VM-exit control is 1, the following must hold for the indicated fields:
 - IA32_FRED_CONFIG: Bit 2, bits 5:4, and bit 11 of the field must be zero. The upper bits of the field must be such that the field’s value is CPU canonical (see Section 4.5).
 - IA32_FRED_RSP1–IA32_FRED_RSP3: The value of each of these fields must be CPU canonical, and bits 5:0 of each field must be zero.
 - IA32_FRED_SSP1–IA32_FRED_SSP3: The value of each of these fields must be CPU canonical, and bits 2:0 of each field must be zero.

29.2.3 Checks on Host Segment and Descriptor-Table Registers

The following checks are performed on fields in the host-state area that correspond to segment and descriptor-table registers:

- In the selector field for each of CS, SS, DS, ES, FS, GS, and TR, the RPL (bits 1:0) and the TI flag (bit 2) must be 0.
- The selector fields for CS and TR cannot be 0000H.
- The selector field for SS cannot be 0000H if the “host address-space size” VM-exit control is 0.
- On processors that support Intel 64 architecture, the base-address fields for FS, GS, GDTR, IDTR, and TR must contain canonical addresses.

1. The bits corresponding to CR0.NW (bit 29) and CR0.CD (bit 30) are never checked because the values of these bits are not changed by VM exit; see Section 30.5.1.

1. Bit 63 of the CR3 field in the host-state area must be 0. This is true even though, If CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 is used to determine whether cached translation information is invalidated.

29.2.4 Checks Related to Address-Space Size

On processors that support Intel 64 architecture, the following checks related to address-space size are performed on VMX controls and fields in the host-state area:

- If the logical processor is outside IA-32e mode (if `IA32_EFER.LMA = 0`) at the time of VM entry, the following must hold:
 - The “IA-32e mode guest” VM-entry control is 0.
 - The “host address-space size” VM-exit control is 0.
- If the logical processor is in IA-32e mode (if `IA32_EFER.LMA = 1`) at the time of VM entry, the “host address-space size” VM-exit control must be 1.
- If the “host address-space size” VM-exit control is 0, the following must hold:
 - The “IA-32e mode guest” VM-entry control is 0.
 - Bit 17 of the CR4 field (corresponding to `CR4.PCIDE`) is 0.
 - Bit 32 of the CR4 field (corresponding to `CR4.FRED`) is 0.
 - Bits 63:32 in the RIP field are 0.
 - If the “load CET state” VM-exit control is 1, bits 63:32 in the `IA32_S_CET` field and in the SSP field are 0.
- If the “host address-space size” VM-exit control is 1, the following must hold:
 - Bit 5 of the CR4 field (corresponding to `CR4.PAE`) is 1.
 - The RIP field contains a canonical address.
 - If the “load CET state” VM-exit control is 1, the `IA32_S_CET` field and the SSP field contain canonical addresses.
- If the “load CET state” VM-exit control is 1, the `IA32_INTERRUPT_SSP_TABLE_ADDR` field contains a canonical address.

On processors that do not support Intel 64 architecture, checks are performed to ensure that the “IA-32e mode guest” VM-entry control and the “host address-space size” VM-exit control are both 0.

29.3 CHECKING AND LOADING GUEST STATE

If all checks on the VMX controls and the host-state area pass (see Section 29.2), the following operations take place concurrently: (1) the guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures; (2) processor state is loaded from the guest-state area or as specified by the VM-entry control fields; and (3) address-range monitoring is cleared.

Because the checking and the loading occur concurrently, a failure may be discovered only after some state has been loaded. For this reason, the logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 29.8.

29.3.1 Checks on the Guest State Area

This section describes checks performed on fields in the guest-state area. These checks may be performed in any order. Some checks prevent establishment of settings (or combinations of settings) that are currently reserved. Future processors may allow such settings (or combinations) and may not perform the corresponding checks. The correctness of software should not rely on VM-entry failures resulting from the checks documented in this section.

The following subsections reference fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

29.3.1.1 Checks on Guest Control Registers, Debug Registers, and MSRs

The following checks are performed on fields in the guest-state area corresponding to control registers, debug registers, and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 26.8). The following are exceptions:
 - Bit 0 (corresponding to CR0.PE) and bit 31 (PG) are not checked if the “unrestricted guest” VM-execution control is 1.¹
 - Bit 29 (corresponding to CR0.NW) and bit 30 (CD) are never checked because the values of these bits are not changed by VM entry; see Section 29.3.2.1.
- If bit 31 in the CR0 field (corresponding to PG) is 1, bit 0 in that field (PE) must also be 1.²
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 26.8).
- If bit 23 in the CR4 field (corresponding to CET) is 1, bit 16 in the CR0 field (WP) must also be 1.
- If the “load debug controls” VM-entry control is 1, bits reserved in the IA32_DEBUGCTL MSR must be 0 in the field for that register. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally.
- The following checks are performed on processors that support Intel 64 architecture:
 - If the “IA-32e mode guest” VM-entry control is 1, bit 31 in the CR0 field (corresponding to CR0.PG) and bit 5 in the CR4 field (corresponding to CR4.PAE) must each be 1.³
 - If the “IA-32e mode guest” VM-entry control is 0, bit 17 in the CR4 field (corresponding to CR4.PCIDE) must be 0.
 - If the “IA-32e mode guest” VM-entry control is 0, bit 32 in the CR4 field (corresponding to CR4.FRED) must be 0.
 - The CR3 field must be such that bits reserved in CR3 are 0.⁴ In particular, the field’s width is limited by MAXPHYADDR, a value derived from the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If the “enable EPT” VM-execution control is 0 (implying that the logical processor is not in SEAM) and IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36]. The CR3 field should not set any of bits 63:MAXPHYADDR.
 - If the “load debug controls” VM-entry control is 1, bits 63:32 in the DR7 field must be 0. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally (if they supported Intel 64 architecture).
 - The IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
 - If the “load CET state” VM-entry control is 1, the IA32_S_CET field and the IA32_INTERRUPT_SSP_TABLE_ADDR field must contain canonical addresses.
- If the “load IA32_PERF_GLOBAL_CTRL” VM-entry control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 22-3).
- If the “load IA32_PAT” VM-entry control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the “load IA32_EFER” VM-entry control is 1, the following checks are performed on the field for the IA32_EFER MSR:

-
1. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 27.6.2.
 2. If the capability MSR IA32_VMX_CRO_FIXED0 reports that CR0.PE must be 1 in VMX operation, bit 0 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
 3. If the capability MSR IA32_VMX_CRO_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
 4. Bit 63 of the CR3 field in the host-state area must be 0. This is true even though, if CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 is used to determine whether cached translation information is invalidated.

- Bits reserved in the IA32_EFER MSR must be 0.
- Bit 10 (corresponding to IA32_EFER.LMA) must equal the value of the “IA-32e mode guest” VM-entry control. It must also be identical to bit 8 (LME) if bit 31 in the CR0 field (corresponding to CR0.PG) is 1.¹
- If the “load IA32_BNDCFGS” VM-entry control is 1, the following checks are performed on the field for the IA32_BNDCFGS MSR:
 - Bits reserved in the IA32_BNDCFGS MSR must be 0.
 - The linear address in bits 63:12 must be canonical.
- If the “load IA32_RTIT_CTL” VM-entry control is 1, bits reserved in the IA32_RTIT_CTL MSR must be 0 in the field for that register (see Table 36-6).
- If the “load CET state” VM-entry control is 1, the IA32_S_CET field must not set any bits reserved in the IA32_S_CET MSR, and bit 10 (corresponding to SUPPRESS) and bit 11 (TRACKER) of the field cannot both be set.
- If the “load guest IA32_LBR_CTL” VM-entry control is 1, bits reserved in the IA32_LBR_CTL MSR must be 0 in the field for that register.
- If the “load PKRS” VM-entry control is 1, bits 63:32 must be 0 in the IA32_PKRS field.
- If the “load UINV” VM-entry control is 1, bits 15:8 must be 0 in the guest UINV field.
- If the “load FRED” VM-entry control is 1, the following must hold for the indicated fields:
 - IA32_FRED_CONFIG: Bit 2, bits 5:4, and bit 11 of the field must be zero. The upper bits of the field must be such that the field’s value is CPU canonical (see Section 4.5).
 - IA32_FRED_RSP1–IA32_FRED_RSP3: The value of each of these fields must be CPU canonical, and bits 5:0 of each field must be zero.
 - IA32_FRED_SSP1–IA32_FRED_SSP3: The value of each of these fields must be CPU canonical, and bits 2:0 of each field must be zero.

29.3.1.2 Checks on Guest Segment Registers

This section specifies the checks on the fields for CS, SS, DS, ES, FS, GS, TR, and LDTR. The following terms are used in defining these checks:

- The guest will be **virtual-8086** if the VM flag (bit 17) is 1 in the RFLAGS field in the guest-state area.
- The guest will be **IA-32e mode** if the “IA-32e mode guest” VM-entry control is 1. (This is possible only on processors that support Intel 64 architecture.)
- The guest will **use FRED transitions** if the FRED bit (bit 32) is 1 in the CR4 field in the guest-state area.
- Any one of these registers is said to be **usable** if the unusable bit (bit 16) is 0 in the access-rights field for that register.

The following are the checks on these fields:

- Selector fields.
 - TR. The TI flag (bit 2) must be 0.
 - LDTR. If LDTR is usable, the TI flag (bit 2) must be 0.
 - SS. If the guest will not be virtual-8086 and the “unrestricted guest” VM-execution control is 0, the RPL (bits 1:0) must equal the RPL of the selector field for CS.²

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

2. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 27.6.2.

- Base-address fields.
 - CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the address must be the selector field shifted left 4 bits (multiplied by 16).
 - The following checks are performed on processors that support Intel 64 architecture:
 - TR, FS, GS. The address must be canonical.
 - LDTR. If LDTR is usable, the address must be canonical.
 - CS. Bits 63:32 of the address must be zero.
 - SS, DS, ES. If the register is usable, bits 63:32 of the address must be zero.
- Limit fields for CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the field must be 0000FFFFH.
- Access-rights fields.
 - CS, SS, DS, ES, FS, GS.
 - If the guest will be virtual-8086, the field must be 000000F3H. This implies the following:
 - Bits 3:0 (Type) must be 3, indicating an expand-up read/write accessed data segment.
 - Bit 4 (S) must be 1.
 - Bits 6:5 (DPL) must be 3.
 - Bit 7 (P) must be 1.
 - Bits 11:8 (reserved), bit 12 (software available), bit 13 (reserved/L), bit 14 (D/B), bit 15 (G), bit 16 (unusable), and bits 31:17 (reserved) must all be 0.
 - If the guest will not be virtual-8086, the different sub-fields are considered separately:
 - Bits 3:0 (Type).
 - CS. The values allowed depend on the setting of the “unrestricted guest” VM-execution control:
 - If the control is 0, the Type must be 9, 11, 13, or 15 (accessed code segment).
 - If the control is 1, the Type must be either 3 (read/write accessed expand-up data segment) or one of 9, 11, 13, and 15 (accessed code segment).
 - SS. If SS is usable, the Type must be 3 or 7 (read/write, accessed data segment).
 - DS, ES, FS, GS. The following checks apply if the register is usable:
 - Bit 0 of the Type must be 1 (accessed).
 - If bit 3 of the Type is 1 (code segment), then bit 1 of the Type must be 1 (readable).
 - Bit 4 (S). If the register is CS or if the register is usable, S must be 1.
 - Bits 6:5 (DPL).
 - CS.
 - If the Type is 3 (read/write accessed expand-up data segment), the DPL must be 0. The Type can be 3 only if the “unrestricted guest” VM-execution control is 1.
 - If the Type is 9 or 11 (non-conforming code segment), the DPL must equal the DPL in the access-rights field for SS.
 - If the Type is 13 or 15 (conforming code segment), the DPL cannot be greater than the DPL in the access-rights field for SS.
 - SS.
 - If the “unrestricted guest” VM-execution control is 0, the DPL must equal the RPL from the selector field.
 - The DPL must be 0 either if the Type in the access-rights field for CS is 3 (read/write accessed expand-up data segment) or if bit 0 in the CR0 field (corresponding to CR0.PE) is 0.¹

- The following checks apply if the guest will use FRED transitions:
 - The DPL must be 0 or 3.
 - If the DPL is 0, the L bit in the CS access-rights field (bit 13) must be 1.
 - If the DPL is 3, the IOPL value in the RFLAGS field (bits 13:12) and the “blocking by STI” bit in the interruptibility-state field (bit 0) must both be 0.
- DS, ES, FS, GS. The DPL cannot be less than the RPL in the selector field if (1) the “unrestricted guest” VM-execution control is 0; (2) the register is usable; and (3) the Type in the access-rights field is in the range 0 – 11 (data segment or non-conforming code segment).
- Bit 7 (P). If the register is CS or if the register is usable, P must be 1.
- Bits 11:8 (reserved). If the register is CS or if the register is usable, these bits must all be 0.
- Bit 14 (D/B). For CS, D/B must be 0 if the guest will be IA-32e mode and the L bit (bit 13) is 1.
- Bit 15 (G). The following checks apply if the register is CS or if the register is usable:
 - If any bit in the limit field in the range 11:0 is 0, G must be 0.
 - If any bit in the limit field in the range 31:20 is 1, G must be 1.
- Bits 31:17 (reserved). If the register is CS or if the register is usable, these bits must all be 0.
- TR. The different sub-fields are considered separately:
 - Bits 3:0 (Type).
 - If the guest will not be IA-32e mode, the Type must be 3 (16-bit busy TSS) or 11 (32-bit busy TSS).
 - If the guest will be IA-32e mode, the Type must be 11 (64-bit busy TSS).
 - Bit 4 (S). S must be 0.
 - Bit 7 (P). P must be 1.
 - Bits 11:8 (reserved). These bits must all be 0.
 - Bit 15 (G).
 - If any bit in the limit field in the range 11:0 is 0, G must be 0.
 - If any bit in the limit field in the range 31:20 is 1, G must be 1.
 - Bit 16 (Unusable). The unusable bit must be 0.
 - Bits 31:17 (reserved). These bits must all be 0.
- LDTR. The following checks on the different sub-fields apply only if LDTR is usable:
 - Bits 3:0 (Type). The Type must be 2 (LDT).
 - Bit 4 (S). S must be 0.
 - Bit 7 (P). P must be 1.
 - Bits 11:8 (reserved). These bits must all be 0.
 - Bit 15 (G).
 - If any bit in the limit field in the range 11:0 is 0, G must be 0.
 - If any bit in the limit field in the range 31:20 is 1, G must be 1.
 - Bits 31:17 (reserved). These bits must all be 0.

1. The following apply if either the “unrestricted guest” VM-execution control or bit 31 of the primary processor-based VM-execution controls is 0: (1) bit 0 in the CR0 field must be 1 if the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation; and (2) the Type in the access-rights field for CS cannot be 3.

29.3.1.3 Checks on Guest Descriptor-Table Registers

The following checks are performed on the fields for GDTR and IDTR:

- On processors that support Intel 64 architecture, the base-address fields must contain canonical addresses.
- Bits 31:16 of each limit field must be 0.

29.3.1.4 Checks on Guest RIP, RFLAGS, and SSP

The following checks are performed on fields in the guest-state area corresponding to RIP, RFLAGS, and SSP (shadow-stack pointer):

- RIP. The following checks are performed on processors that support Intel 64 architecture:
 - Bits 63:32 must be 0 if the “IA-32e mode guest” VM-entry control is 0 or if the L bit (bit 13) in the access-rights field for CS is 0.
 - If the processor supports $N < 64$ linear-address bits, bits 63:N must be identical if the “IA-32e mode guest” VM-entry control is 1 and the L bit in the access-rights field for CS is 1.¹ (No check applies if the processor supports 64 linear-address bits.) The guest RIP value is not required to be canonical; the value of bit N-1 may differ from that of bit N.
- RFLAGS.
 - Reserved bits 63:22 (bits 31:22 on processors that do not support Intel 64 architecture), bit 15, bit 5 and bit 3 must be 0 in the field, and reserved bit 1 must be 1.
 - The VM flag (bit 17) must be 0 either if the “IA-32e mode guest” VM-entry control is 1 or if bit 0 in the CR0 field (corresponding to CR0.PE) is 0.²
 - The IF flag (RFLAGS[bit 9]) must be 1 if the valid bit (bit 31) in the [injected-event identification](#) field is 1 and the [event](#) type (bits 10:8) is external interrupt.
- SSP. The following checks are performed if the “load CET state” VM-entry control is 1
 - Bits 1:0 must be 0.
 - If the processor supports the Intel 64 architecture, bits 63:N must be identical, where N is the CPU’s maximum linear-address width. (This check does not apply if the processor supports 64 linear-address bits.) The guest SSP value is not required to be canonical; the value of bit N-1 may differ from that of bit N.

29.3.1.5 Checks on Guest Non-Register State

The following checks are performed on fields in the guest-state area corresponding to non-register state:

- Activity state.
 - The activity-state field must contain a value in the range 0 – 3, indicating an activity state supported by the implementation (see Section •). Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine what activity states are supported.
 - The activity-state field must not indicate the HLT state if the DPL (bits 6:5) in the access-rights field for SS is not 0.³
 - The activity-state field must indicate the active state if the interruptibility-state field indicates blocking by either MOV-SS or by STI (if either bit 0 or bit 1 in that field is 1).
 - If the valid bit (bit 31) in the [injected-event identification](#) field is 1, the [event](#) to be delivered (as defined by [event](#) type and vector) must not be one that would normally be blocked while a logical processor is in the

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.
2. If the capability MSR IA32_VMX_CRO_FIXED0 reports that CR0.PE must be 1 in VMX operation, bit 0 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
3. As noted in Section 27.4.1, SS.DPL corresponds to the logical processor’s current privilege level (CPL).

activity state corresponding to the contents of the activity-state field. The following items enumerate the **events** (as specified in the **injected-event identification** field) whose injection is allowed for the different activity states:

- Active. Any **event** is allowed.
- HLT. The only events allowed are the following:
 - Those with **event** type external interrupt or non-maskable interrupt (NMI).
 - Those with **event** type hardware exception and vector 1 (debug exception) or vector 18 (machine-check exception).
 - Those with **event** type other event and vector 0 (pending MTF VM exit).

See Table 27-18 in Section 27.8.3 for details regarding the format of the **injected-event identification** field.

- Shutdown. Only NMIs and machine-check exceptions are allowed.
- Wait-for-SIPI. No **events** are allowed.

— The activity-state field must not indicate the wait-for-SIPI state if the “entry to SMM” VM-entry control is 1.

- Interruptibility state.

- The reserved bits (bits 31:5) must be 0.
- The field cannot indicate blocking by both STI and MOV SS (bits 0 and 1 cannot both be 1).
- Bit 0 (blocking by STI) must be 0 if the IF flag (bit 9) is 0 in the RFLAGS field.
- Bit 0 (blocking by STI) and bit 1 (blocking by MOV-SS) must both be 0 if the valid bit (bit 31) in the **injected-event identification** field is 1 and the **event** type (bits 10:8) in that field has value 0, indicating external interrupt, or value 2, indicating non-maskable interrupt (NMI).
- Bit 2 (blocking by SMI) must be 0 if the processor is not in SMM.
- Bit 2 (blocking by SMI) must be 1 if the “entry to SMM” VM-entry control is 1.
- Bit 3 (blocking by NMI) must be 0 if the “virtual NMIs” VM-execution control is 1, the valid bit (bit 31) in the **injected-event identification** field is 1, and the **event** type (bits 10:8) in that field has value 2 (indicating NMI).
- If bit 4 (enclave interruption) is 1, bit 1 (blocking by MOV-SS) must be 0 and the processor must support for SGX by enumerating CPUID.07H.00H:EBX.SGX[2] as 1.

NOTE

If the “virtual NMIs” VM-execution control is 0, there is no requirement that bit 3 be 0 if the valid bit in the **injected-event identification** field is 1 and the **event** type in that field has value 2.

- Pending debug exceptions.

- Bits 11:4, bit 13, bit 15, and bits 63:17 (bits 31:17 on processors that do not support Intel 64 architecture) must be 0.
- The following checks are performed if any of the following holds: (1) the interruptibility-state field indicates blocking by STI (bit 0 in that field is 1); (2) the interruptibility-state field indicates blocking by MOV SS (bit 1 in that field is 1); or (3) the activity-state field indicates HLT:
 - Bit 14 (BS) must be 1 if the TF flag (bit 8) in the RFLAGS field is 1 and the BTF flag (bit 1) in the IA32_DEBUGCTL field is 0.
 - Bit 14 (BS) must be 0 if the TF flag (bit 8) in the RFLAGS field is 0 or the BTF flag (bit 1) in the IA32_DEBUGCTL field is 1.
- The following checks are performed if bit 16 (RTM) is 1:
 - Bits 11:0, bits 15:13, and bits 63:17 (bits 31:17 on processors that do not support Intel 64 architecture) must be 0; bit 12 must be 1.

- The processor must support for RTM by enumerating CPUID.07H.00H:EBX[11] as 1.
- The interruptibility-state field must not indicate blocking by MOV SS (bit 1 in that field must be 0).
- VMCS link pointer. The following checks apply if the field contains a value other than FFFFFFFF_FFFFFFFFH:
 - Bits 11:0 must be 0.
 - The field's width is limited by MAXPHYADDR, a value derived from the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM. The VMCS link pointer should not set bits in the range 63:MAXPHYADDR.
 - The 4 bytes located in memory referenced by the value of the field (as a physical address) must satisfy the following:
 - Bits 30:0 must contain the processor's VMCS revision identifier (see Section 27.2).¹
 - Bit 31 must contain the setting of the "VMCS shadowing" VM-execution control.² This implies that the referenced VMCS is a shadow VMCS (see Section 27.10) if and only if the "VMCS shadowing" VM-execution control is 1.
 - If the processor is not in SMM or the "entry to SMM" VM-entry control is 1, the field must not contain the current VMCS pointer.
 - If the processor is in SMM and the "entry to SMM" VM-entry control is 0, the field must differ from the executive-VMCS pointer.

29.3.1.6 Checks on Guest Page-Directory-Pointer-Table Entries

If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0, the logical processor uses **PAE paging** (see Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A).³ When PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTes). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTes.

A VM entry is to a guest that uses PAE paging if (1) bit 31 (corresponding to CR0.PG) is set in the CR0 field in the guest-state area; (2) bit 5 (corresponding to CR4.PAE) is set in the CR4 field; and (3) the "IA-32e mode guest" VM-entry control is 0. Such a VM entry checks the validity of the PDPTes:

- If the "enable EPT" VM-execution control is 0, VM entry checks the validity of the PDPTes referenced by the CR3 field in the guest-state area if either (1) PAE paging was not in use before the VM entry; or (2) the value of CR3 is changing as a result of the VM entry. VM entry may check their validity even if neither (1) nor (2) hold.⁴
- If the "enable EPT" VM-execution control is 1, VM entry checks the validity of the PDPTe fields in the guest-state area (see Section •).

A VM entry to a guest that does not use PAE paging does not check the validity of any PDPTes.

A VM entry that checks the validity of the PDPTes uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use.⁵ If MOV to CR3 would cause a general-protection exception due to the PDPTes that would be loaded (e.g., because a reserved bit is set), the VM entry fails.

-
1. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field. For all processors produced prior to this change, bit 31 of the VMCS revision identifier was 0.
 2. "VMCS shadowing" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "VMCS shadowing" VM-execution control were 0. See Section 27.6.2.
 3. On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine the number physical-address bits supported by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
 4. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "enable EPT" VM-execution control were 0. See Section 27.6.2.
 5. This implies that (1) bits 11:9 in each PDPTe are ignored; and (2) if bit 0 (present) is clear in one of the PDPTes, bits 63:1 of that PDPTe are ignored.

29.3.2 Loading Guest State

Processor state is updated on VM entries in the following ways:

- Some state is loaded from the guest-state area.
- Some state is determined by VM-entry controls.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order and in parallel with the checking of VMCS contents (see Section 29.3.1).

The loading of guest state is detailed in Section 29.3.2.1 to Section 29.3.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

In addition to the state loading described in this section, VM entries may load MSRs from the VM-entry MSR-load area (see Section 29.4). This loading occurs only after the state loading described in this section and the checking of VMCS contents described in Section 29.3.1.

29.3.2.1 Loading Guest Control Registers, Debug Registers, and MSRs

The following items describe how guest control registers, debug registers, and MSRs are loaded on VM entry:

- CR0 is loaded from the CR0 field with the exception of the following bits, which are never modified on VM entry: ET (bit 4); reserved bits 15:6, 17, and 28:19; NW (bit 29) and CD (bit 30).¹ The values of these bits in the CR0 field are ignored.
- CR3 and CR4 are loaded from the CR3 field and the CR4 field, respectively.
- If the “load debug controls” VM-entry control is 1, DR7 is loaded from the DR7 field with the exception that bit 12 and bits 15:14 are always 0 and bit 10 is always 1. The values of these bits in the DR7 field are ignored. The first processors to support the virtual-machine extensions supported only the 1-setting of the “load debug controls” VM-entry control and thus always loaded DR7 from the DR7 field.
- The following describes how certain MSRs are loaded using fields in the guest-state area:
 - If the “load debug controls” VM-entry control is 1, the IA32_DEBUGCTL MSR is loaded from the IA32_DEBUGCTL field. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always loaded the IA32_DEBUGCTL MSR from the IA32_DEBUGCTL field.
 - The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since this field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
 - The IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
 - The following are performed on processors that support Intel 64 architecture:
 - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 29.3.2.2).
 - If the “load IA32_EFER” VM-entry control is 0, bits in the IA32_EFER MSR are modified as follows:
 - IA32_EFER.LMA is loaded with the setting of the “IA-32e mode guest” VM-entry control.
 - If CR0 is being loaded so that CR0.PG = 1, IA32_EFER.LME is also loaded with the setting of the “IA-32e mode guest” VM-entry control.² Otherwise, IA32_EFER.LME is unmodified.

See below for the case in which the “load IA32_EFER” VM-entry control is 1

1. Bits 15:6, bit 17, and bit 28:19 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. Bits 15:6, bit 17, and bit 28:19 of CR0 are always 0 and CR0.ET is always 1.

2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, VM entry must be loading CR0 so that CR0.PG = 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- If the “load IA32_PERF_GLOBAL_CTRL” VM-entry control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field.
- If the “load IA32_PAT” VM-entry control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field.
- If the “load IA32_EFER” VM-entry control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field.
- If the “load IA32_BNDCFGS” VM-entry control is 1, the IA32_BNDCFGS MSR is loaded from the IA32_BNDCFGS field.
- If the “load IA32_RTIT_CTL” VM-entry control is 1, the IA32_RTIT_CTL MSR is loaded from the IA32_RTIT_CTL field.
- If the “load CET” VM-entry control is 1, the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs are loaded from the IA32_S_CET field and the IA32_INTERRUPT_SSP_TABLE_ADDR field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
- If the “load guest IA32_LBR_CTL” VM-entry control is 1, the IA32_LBR_CTL MSR is loaded from the IA32_LBR_CTL guest state field.
- If the “load PKRS” VM-entry control is 1, the IA32_PKRS MSR is loaded from the IA32_PKRS field.
- If the “load UINV” VM-entry control is 1, UINV is loaded with the low 8 bits of the UINV field. UINV is represented in bits 39:32 of the IA32_UINTR_MISC MSR. The remainder of the MSR is not modified.
- If the “load FRED” VM-entry control is 1, the following MSRs are loaded from the corresponding fields: IA32_FRED_CONFIG, IA32_FRED_RSP1, IA32_FRED_RSP2, IA32_FRED_RSP3, IA32_FRED_STKLVLs, IA32_FRED_SSP1, IA32_FRED_SSP2, and IA32_FRED_SSP3.

With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-entry MSR-load area. See Section 29.4.

- The SMBASE register is unmodified by all VM entries except those that return from SMM.

29.3.2.2 Loading Guest Segment Registers and Descriptor-Table Registers

For each of CS, SS, DS, ES, FS, GS, TR, and LDTR, fields are loaded from the guest-state area as follows:

- The unusable bit is loaded from the access-rights field. This bit can never be set for TR (see Section 29.3.1.2). If it is set for one of the other registers, the following apply:
 - For each of CS, SS, DS, ES, FS, and GS, uses of the segment cause faults (general-protection exception or stack-fault exception) outside 64-bit mode, just as they would had the segment been loaded using a null selector. This bit does not cause accesses to fault in 64-bit mode.
 - If this bit is set for LDTR, uses of LDTR cause general-protection exceptions in all modes, just as they would had LDTR been loaded using a null selector.
- If this bit is clear for any of CS, SS, DS, ES, FS, GS, TR, and LDTR, a null selector value does not cause a fault (general-protection exception or stack-fault exception).
- TR. The selector, base, limit, and access-rights fields are loaded.
- CS.
 - The following fields are always loaded: selector, base address, limit, and (from the access-rights field) the L, D, and G bits.
 - For the other fields, the unusable bit of the access-rights field is consulted:
 - If the unusable bit is 0, all of the access-rights field is loaded.
 - If the unusable bit is 1, the remainder of CS access rights are undefined after VM entry.
- SS, DS, ES, FS, GS, and LDTR.
 - The selector fields are loaded.
 - For the other fields, the unusable bit of the corresponding access-rights field is consulted:
 - If the unusable bit is 0, the base-address, limit, and access-rights fields are loaded.

- If the unusable bit is 1, the base address, the segment limit, and the remainder of the access rights are undefined after VM entry with the following exceptions:
 - Bits 3:0 of the base address for SS are cleared to 0.
 - SS.DPL is always loaded from the SS access-rights field. This will be the current privilege level (CPL) after the VM entry completes.
 - SS.B is always set to 1.
 - The base addresses for FS and GS are loaded from the corresponding fields in the VMCS. On processors that support Intel 64 architecture, the values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.
 - On processors that support Intel 64 architecture, bits 63:32 of the base addresses for SS, DS, and ES are cleared to 0.

GDTR and IDTR are loaded using the base and limit fields.

29.3.2.3 Loading Guest RIP, RSP, RFLAGS, and SSP

RSP, RIP, and RFLAGS are loaded from the RSP field, the RIP field, and the RFLAGS field, respectively.

If the “load CET” VM-entry control is 1, SSP (shadow-stack pointer) is loaded from the SSP field.

The following items regard the upper 32 bits of these fields on VM entries that are not to 64-bit mode:

- Bits 63:32 of RSP are undefined outside 64-bit mode. Thus, a logical processor may ignore the contents of bits 63:32 of the RSP field on VM entries that are not to 64-bit mode.
- As noted in Section 29.3.1.4, bits 63:32 of the RIP and RFLAGS fields must be 0 on VM entries that are not to 64-bit mode. (The same is true for SSP for VM entries that are not to 64-bit mode when the “load CET” VM-entry control is 1.)

29.3.2.4 Loading Page-Directory-Pointer-Table Entries

As noted in Section 29.3.1.6, the logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0. A VM entry to a guest that uses PAE paging loads the PDPTs into internal, non-architectural registers based on the setting of the “enable EPT” VM-execution control:

- If the control is 0, the PDPTs are loaded from the page-directory-pointer table referenced by the physical address in the value of CR3 being loaded by the VM entry (see Section 29.3.2.1). The values loaded are treated as physical addresses in VMX non-root operation.
- If the control is 1, the PDPTs are loaded from corresponding fields in the guest-state area (see Section •). The values loaded are treated as guest-physical addresses in VMX non-root operation.

29.3.2.5 Updating Non-Register State

Section 31.4 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM entries invalidate cached mappings:

- If the “enable VPID” VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EPTRTA values (EPTRTA is the value of bits 51:12 of EPTP).
- VM entries are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

If the “virtual-interrupt delivery” VM-execution control is 1, VM entry loads the values of RVI and SVI from the guest interrupt-status field in the VMCS (see Section •). After doing so, the logical processor first causes PPR virtualization (Section 32.1.3) and then evaluates pending virtual interrupts (Section 32.2.1).

If a virtual interrupt is recognized, it may be delivered in VMX non-root operation immediately after VM entry (including any specified event injection) completes; see Section 29.7.5. See Section 32.2.2 for details regarding the delivery of virtual interrupts.

29.3.3 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 11.10.4 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A. VM entries clear any address-range monitoring that may be in effect.

29.4 LOADING MSRS

VM entries may load MSRs from the VM-entry MSR-load area (see Section 27.8.2). Specifically each entry in that area (up to the number specified in the VM-entry MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.¹

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101 (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM entry did not commence in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be loaded on VM entries for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.
- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.²

The VM entry fails if processing fails for any entry. The logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 29.8.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM entry, the logical processor will not use any translations that were cached before the transition.

29.5 TRACE-ADDRESS PRE-TRANSLATION (TAPT)

When the "Intel PT uses guest physical addresses" VM-execution control is 1, the addresses used by Intel PT are treated as guest-physical addresses, and these are translated to physical addresses using EPT.

VM entry uses **trace-address pre-translation (TAPT)** to prevent buffered trace data from being lost due to an EPT violation; see Section 28.5.4.2. VM entry uses TAPT only if Intel PT will be enabled following VM entry (IA32_RTIT_CTL.TraceEn = 1) and only if the "Intel PT uses guest physical addresses" VM-execution control is 1.

As noted in Section 28.5.4, TAPT may cause a VM exit due to an EPT violation, EPT misconfiguration, page-modification log-full event, or APIC access. If such a VM exit occurs as a result of TAPT during VM entry, the VM exit operates as if it had occurred in VMX non-root operation after the VM entry completed (in the guest context).

If TAPT during VM entry causes a VM exit, the VM entry does not perform event injection (Section 29.6), even if the valid bit in the **injected-event identification** field is 1. Such VM exits save the contents of **injected-event identification** and **injected-event** error code fields into the **original-event identification** and **original-event** error code fields, respectively.

1. Because attempts to modify the value of IA32_EFER.LMA by WRMSR are ignored, attempts to modify it using the VM-entry MSR-load area are also ignored.

2. If CR0.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. If VM entry has established CR0.PG = 1, the IA32_EFER MSR should not be included in the VM-entry MSR-load area for the purpose of modifying the LME bit.

29.6 EVENT INJECTION

If the valid bit in the **injected-event identification** field (see Section 27.8.3) is 1, VM entry causes an event to be delivered (or, in one case, made pending) after all components of guest state have been loaded (including MSRs) and after the VM-execution control fields have been established.

- If the **event** type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception), the event is delivered as described in Section 29.6.1. This is done also if the event type is 7 (other event) and the vector field is 1 (SYSCALL) or 2 (SYSENTER).
- If the **event** type in the field is 7 (other event) and the vector field is 0, an MTF VM exit is pending after VM entry. See Section 29.6.2.

29.6.1 Injection of Interrupts and Exceptions

VM entry delivers an injected event within the guest context established by VM entry. This means that delivery occurs after all components of guest state have been loaded (including MSRs) and after the VM-execution control fields have been established:¹

- If **FRED transitions are not enabled**, the event is delivered using the vector in that field to select a descriptor in the IDT. (Since event injection occurs after loading IDTR from the guest-state area, this is the guest IDT.)
- If **FRED transitions are enabled**, the event is delivered using **FRED event delivery**. (Since event injection occurs after any optional loading of the MSRs that control FRED transitions, those MSRs control this event delivery.)

Section 29.6.1.1 provides details of event injection. In general, the event is delivered exactly as if it had been generated normally.

An exception is made if the following all hold: bit 25 (UINTR) is set to 1 in the guest CR4 field and the “IA-32e mode guest” VM-entry control is 1, and VM entry is injecting an external interrupt whose vector is the value that UINV would have after VM entry. In this case, the logical processor **does not deliver the interrupt but instead** performs user-interrupt notification processing as specified in Section 9.5.2. (If the guest activity-state field indicated the HLT state, the logical processor enters the HLT state following user-interrupt notification processing.)

If event delivery (or user-interrupt notification processing; see above) encounters a nested exception (for example, a general-protection exception because the vector indicates a descriptor beyond the IDT limit), the exception bitmap is consulted using the vector of that exception:

- If the bit for the nested exception is 0, the nested exception is **handled** normally. If the nested exception is benign, it is delivered **normally**. If it is contributory or a page fault, a double fault may be generated, depending on the nature of the event whose delivery encountered the nested exception. See Chapter 7, “Interrupt 8—Double Fault Exception (#DF)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.²
- If the bit for the nested exception is 1, a VM exit occurs. Section 29.6.1.2 details cases in which event injection causes a VM exit.

29.6.1.1 Details of Event Injection

The event-injection process is controlled by the contents of the **injected-event identification** field (format given in Table 27-18), the **injected-event** error-code field, the **injected-event data field (for FRED event delivery)**, and the VM-entry instruction-length field. The following items provide details of the process:

- The value pushed on the stack for RFLAGS is generally that which was loaded from the guest-state area. The value pushed for the RF flag is not modified based on the type of event being delivered. However, the pushed

1. This does not imply that injection of an exception or interrupt will cause a VM exit due to the settings of VM-execution control fields (such as the exception bitmap) that would cause a VM exit if the event had occurred in VMX non-root operation. In contrast, a nested exception encountered during event delivery may cause a VM exit; see Section 29.6.1.1.

2. Hardware exceptions with the following unused vectors are considered benign: 15 and 21–31. A hardware exception with vector 20 is considered benign unless the processor supports the 1-setting of the “EPT-violation #VE” VM-execution control; in that case, it has the same severity as page faults.

value of RFLAGS may be modified if a software interrupt is being injected into a guest that will be in virtual-8086 mode (see below). After RFLAGS is pushed on the stack, the value in the RFLAGS register is modified as is done normally when delivering an event.

- The instruction pointer that is pushed on the stack depends on the type of event and whether nested exceptions occur during its delivery. The term **current guest RIP** refers to the value to be loaded from the guest-state area. The value pushed is determined as follows:¹
 - If VM entry successfully injects (with no nested exception) an event with **event** type external interrupt, NMI, or hardware exception, the current guest RIP is pushed on the stack.
 - If VM entry successfully injects (with no nested exception) an event with **event** type software interrupt, privileged software exception, or software exception, the current guest RIP is incremented by the VM-entry instruction length before being pushed on the stack. **This item applies also if VM entry is injecting SYSCALL or SYSENTER (with FRED event delivery).**
 - If VM entry encounters an exception while injecting an event and that exception does not cause a VM exit, the current guest RIP is pushed on the stack regardless of event type or VM-entry instruction length. If the encountered exception does cause a VM exit that saves RIP, the saved RIP is current guest RIP.
- If the deliver-error-code bit (bit 11) is set in the **injected-event identification** field, the contents of the **injected-event** error-code field is pushed on the stack as an error code would be pushed during delivery of an exception. **If this bit is clear, FRED event delivery pushes zero for an error code.**
- DR6, DR7, and the IA32_DEBUGCTL MSR are not modified by event injection, even if the event has vector 1 (normal deliveries of debug exceptions, which have vector 1, do update these registers).
- If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode (RFLAGS.VM = 1), no general-protection exception can occur due to RFLAGS.IOPL < 3. A VM monitor should check RFLAGS.IOPL before injecting such an event and, if desired, inject a general-protection exception instead of a software interrupt.
- If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode with virtual-8086 mode extensions (RFLAGS.VM = CR4.VME = 1), event delivery is subject to VME-based interrupt redirection based on the software interrupt redirection bitmap in the task-state segment (TSS) as follows:
 - If bit *n* in the bitmap is clear (where *n* is the number of the software interrupt), the interrupt is directed to an 8086 program interrupt handler: the processor uses a 16-bit interrupt-vector table (IVT) located at linear address zero. If the value of RFLAGS.IOPL is less than 3, the following modifications are made to the value of RFLAGS that is pushed on the stack: IOPL is set to 3, and IF is set to the value of VIF.
 - If bit *n* in the bitmap is set (where *n* is the number of the software interrupt), the interrupt is directed to a protected-mode interrupt handler. (In other words, the injection is treated as described in the next item.) In this case, the software interrupt does not invoke such a handler if RFLAGS.IOPL < 3 (a general-protection exception occurs instead). However, as noted above, RFLAGS.IOPL cannot cause an injected software interrupt to cause such a exception. Thus, in this case, the injection invokes a protected-mode interrupt handler independent of the value of RFLAGS.IOPL.

Injection of events of other types are not subject to this redirection.

- If VM entry is injecting a software interrupt (not redirected as described above) or software exception, privilege checking is performed on the IDT descriptor being accessed as would be the case for executions of INT *n*, INT3, or INTO (the descriptor's DPL cannot be less than CPL). There is no checking of RFLAGS.IOPL, even if the guest will be in virtual-8086 mode. Failure of this check may lead to a nested exception. Injection of an event with **event** type external interrupt, NMI, hardware exception, and privileged software exception, or with **event** type software interrupt and being redirected as described above, do not perform these checks.
- If VM entry is injecting a non-maskable interrupt (NMI) and the "virtual NMIs" VM-execution control is 1, virtual-NMI blocking is in effect after VM entry.
- The transition causes a last-branch record to be logged if the LBR bit is set in the IA32_DEBUGCTL MSR. This is true even for events such as debug exceptions, which normally clear the LBR bit before delivery.
- The last-exception record MSRs (LERs) may be updated based on the setting of the LBR bit in the IA32_DEBUGCTL MSR. Events such as debug exceptions, which normally clear the LBR bit before they are delivered, and therefore do not normally update the LERs, may do so as part of VM-entry event injection.

1. While these items refer to RIP, the width of the value pushed (16 bits, 32 bits, or 64 bits) is determined normally.

- If injection of an event encounters a nested exception, the value of the EXT bit (bit 0) in any error code for that nested exception is determined as follows:
 - If event being injected has [event](#) type external interrupt, NMI, hardware exception, or privileged software exception and encounters a nested exception (but does not produce a double fault), the error code for that exception sets the EXT bit.
 - If event being injected is a software interrupt, a software exception, [or \(for FRED event delivery\) SYSCALL or SYSENTER](#), and encounters a nested exception, the error code for that exception clears the EXT bit.
 - If event delivery encounters a nested exception and delivery of that exception encounters another exception (but does not produce a double fault), the error code for that exception sets the EXT bit.
 - If a double fault is produced, the error code for the double fault is 0000H (the EXT bit is clear).
- The following items apply specifically when the injected event is delivered with FRED event delivery (because FRED transitions will be enabled after VM entry):
 - Parts of FRED event delivery depend on the CPL at the time the event occurred. For an event injected by VM entry, the value used for CPL is that which VM entry loaded for DPL in the access-rights field for SS.
 - If bit 13 of the injected-event identification field is 1, FRED event delivery determines stack level as if the event being delivered is an exception that was encountered during delivery of another event. (Otherwise, it does not). FRED event delivery saves the value of this bit for the “nested exception” indication in the event information saved on the stack (specifically, this is bit 58 of the augmented SS field on the stack). This bit is not used for events injected with IDT event delivery.
 - The stack frame established by FRED event delivery includes 64 bits of event data that is defined for certain events stack. For an event injected by VM entry, the event data saved is the value of the injected-event-data field in the VMCS. This is done for all injected events (even those that do not normally define event data).
 - FRED event delivery of certain events saves an instruction length as part of event information. For such an event injected by VM entry, the instruction length saved is the value of the VM-entry instruction-length field.
 - FRED event delivery saves as part of event information a bit that indicates whether event occurred while the logical processor was in enclave mode. For an event injected by VM entry, the value saved is that of bit 4 of the guest interruptibility-state field (which indicates “enclave interruption”).
 - The following items provide details of how FRED event delivery of an event injected by VM entry sets bits in the augmented CS and the augmented SS saved on the stack:¹
 - Bits 17:16 of the augmented CS (stack level) are set to a value that depends on the CPL from which the event is being delivered (see above for how this CPL is determined):
 - If the CPL is 0, bits 17:16 are set to the value that would have been in IA32_FRED_CONFIG[1:0] had VM entry completed without injecting an event.²
 - If the CPL is 3, bits 17:16 are zero.
 - Bit 18 of the augmented CS (indirect branch tracker) is set to a value that depends on the CPL from which the event is being delivered and the value of bit 23 (CET) in the guest CR4 field:
 - If the CPL is 3 or the guest CR4.CET is 0, bit 18 is 0.
 - If the CPL is 0 and the guest CR4.CET is 1, the bit is set to the value of IA32_S_CET.ENDBR AND IA32_S_CET.TRACKER, referring to the value that the IA32_S_CET MSR will have following VM entry.
 - Bit 16 of the augmented SS (blocking by STI) is set to the value of bit 0 of the guest interruptibility-state (that bit indicates blocking by STI). (There is no blocking by STI after event injection.)

1. If supervisor shadow stacks will be enabled, the same augmented CS is pushed on the shadow stack;

2. If the “load FRED” VM-entry control is 1, this will be bits 1:0 of the IA32_FRED_CONFIG field, unless the IA32_FRED_CONFIG MSR was subsequently loaded from the VM-entry MSR-load area (in which case it will be bits 1:0 of the value loaded from memory).

- Bit 17 of the augmented SS (software interrupt or system call) is set if and only if the value of bits 10:8 in the injected-event identification field is 4 (indicating event type software interrupt) or 7 (indicating event type “other event,” when used for SYSCALL and SYSENTER).¹
- Bit 18 of the augmented SS (NMI) is set if and only if the value of bits 10:8 in the injected-event identification field in the VMCS is 2 (indicating event type NMI).

29.6.1.2 VM Exits During Event Injection

An event being injected never causes a VM exit directly regardless of the settings of the VM-execution controls. For example, setting the “NMI exiting” VM-execution control to 1 does not cause a VM exit due to injection of an NMI.

However, the event-delivery process may lead to a VM exit:

- If the vector in the [injected-event identification](#) field identifies a task gate in the IDT, the attempted task switch may cause a VM exit just as it would had the injected event occurred during normal execution in VMX non-root operation (see Section 28.4.2).
- If event delivery encounters a nested exception, a VM exit may occur depending on the contents of the exception bitmap (see Section 28.2).
- If event delivery generates a double-fault exception (due to a nested exception); the logical processor encounters another nested exception while attempting to call the double-fault handler; and that exception does not cause a VM exit due to the exception bitmap; then a VM exit occurs due to triple fault (see Section 28.2).
- If event delivery injects a double-fault exception and encounters a nested exception that does not cause a VM exit due to the exception bitmap, then a VM exit occurs due to triple fault (see Section 28.2).
- If the “virtualize APIC accesses” VM-execution control is 1 and event delivery generates an access to the APIC-access page, that access is treated as described in Section 32.4 and may cause a VM exit.²

If the event-delivery process does cause a VM exit, the processor state before the VM exit is determined just as it would be had the injected event occurred during normal execution in VMX non-root operation. If the injected event directly accesses a task gate that cause a VM exit or if the first nested exception encountered causes a VM exit, information about the injected event is saved in the [original-event identification](#) field (see Section 30.2.4).

The material in this section applies also if injection of an external interrupt results in user-interrupt notification processing instead of event delivery (see Section 29.6.1 earlier).

29.6.1.3 Event Injection for VM Entries to Real-Address Mode

If VM entry is loading CR0.PE with 0, any injected event is delivered as would normally be done in real-address mode.³ Specifically, VM entry uses the vector provided in the [injected-event identification](#) field to select a 4-byte entry from an interrupt-vector table at the linear address in IDTR.base. Further details are provided in Section 16.1.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Because bit 11 (deliver error code) in the [injected-event identification](#) field must be 0 if CR0.PE will be 0 after VM entry (see Section 29.2.1.3), events injected with CR0.PE = 0 do not push an error code on the stack. This is consistent with event delivery in real-address mode.

If event delivery encounters a fault (due to a violation of IDTR.limit or of SS.limit), the fault is treated as if it had occurred during event delivery in VMX non-root operation. Such a fault may lead to a VM exit as discussed in Section 29.6.1.2.

1. Event type 7 may also be used for pending MTF VM exits, but these are not delivered using FRED event delivery.
2. “Virtualize APIC accesses” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “virtualize APIC accesses” VM-execution control were 0. See Section 27.6.2.
3. If the capability MSR IA32_VMX_CRO_FIXED0 reports that CR0.PE must be 1 in VMX operation, VM entry must be loading CR0.PE with 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

29.6.2 Injection of Pending MTF VM Exits

If the **event** type in the **injected-event identification** field is 7 (other event) and the vector field is 0, VM entry causes an MTF VM exit to be pending on the instruction boundary following VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0. See Section 28.5.2 for the treatment of pending MTF VM exits.

29.7 SPECIAL FEATURES OF VM ENTRY

This section details a variety of features of VM entry. It uses the following terminology: a VM entry is **injecting** if the valid bit (bit 31) of the **injected-event identification** is 1 and the **event** type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception).

29.7.1 Interruptibility State

The interruptibility-state field in the guest-state area (see Table 27-3) contains bits that control blocking by STI, blocking by MOV SS, and blocking by NMI. This field impacts event blocking after VM entry as follows:

- If the VM entry is **injecting**, there is no blocking by STI or by MOV SS following the VM entry, regardless of the contents of the interruptibility-state field.
- If the VM entry is not **injecting**, the following apply:
 - Events are blocked by STI if and only if bit 0 in the interruptibility-state field is 1. This blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry; see Section 29.7.3).
 - Events are blocked by MOV SS if and only if bit 1 in the interruptibility-state field is 1. This may affect the treatment of pending debug exceptions; see Section 29.7.3. This blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry).
- The blocking of non-maskable interrupts (NMIs) is determined as follows:
 - If the “virtual NMIs” VM-execution control is 0, NMIs are blocked if and only if bit 3 (blocking by NMI) in the interruptibility-state field is 1. If the “NMI exiting” VM-execution control is 0, execution of the IRET instruction removes this blocking (even if the instruction generates a fault). If the “NMI exiting” control is 1, IRET does not affect this blocking.
 - The following items describe the use of bit 3 (blocking by NMI) in the interruptibility-state field if the “virtual NMIs” VM-execution control is 1:
 - The bit’s value does not affect the blocking of NMIs after VM entry. NMIs are not blocked in VMX non-root operation (except for ordinary blocking for other reasons, such as by the MOV SS instruction, the wait-for-SIPI state, etc.)
 - The bit’s value determines whether there is virtual-NMI blocking after VM entry. If the bit is 1, virtual-NMI blocking is in effect after VM entry. If the bit is 0, there is no virtual-NMI blocking after VM entry unless the VM entry is injecting an NMI (see Section 29.6.1.1). Execution of IRET removes virtual-NMI blocking (even if the instruction generates a fault).

If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0; see Section 29.2.1.1.

- Blocking of system-management interrupts (SMIs) is determined as follows:
 - If the VM entry was not executed in system-management mode (SMM), SMI blocking is unchanged by VM entry.
 - If the VM entry was executed in SMM, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.

29.7.2 Activity State

The activity-state field in the guest-state area controls whether, after VM entry, the logical processor is active or in one of the inactive states identified in Section 27.4.2. The use of this field is determined as follows:

- If the VM entry is **injecting**, the logical processor is in the active state after VM entry. While the consistency checks described in Section 29.3.1.5 on the activity-state field do apply in this case, the contents of the activity-state field do not determine the activity state after VM entry.
- If the VM entry is not **injecting**, the logical processor ends VM entry in the activity state specified in the guest-state area. If VM entry ends with the logical processor in an inactive activity state, the VM entry generates any special bus cycle that is normally generated when that activity state is entered from the active state. If VM entry would end with the logical processor in the shutdown state and the logical processor is in SMX operation,¹ an Intel® TXT shutdown condition occurs. See the Intel® Trusted Execution Technology Measured Launched Environment Developer's Guide. The error code used is 0000H, indicating "legacy shutdown."
- Some activity states unconditionally block certain events. The following blocking is in effect after any VM entry that puts the processor in the indicated state:
 - The active state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the active state and in VMX non-root operation are discarded and do not cause VM exits.
 - The HLT state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the HLT state and in VMX non-root operation are discarded and do not cause VM exits.
 - The shutdown state blocks external interrupts and SIPIs. External interrupts that arrive while a logical processor is in the shutdown state and in VMX non-root operation do not cause VM exits even if the "external-interrupt exiting" VM-execution control is 1. SIPIs that arrive while a logical processor is in the shutdown state and in VMX non-root operation are discarded and do not cause VM exits.
 - The wait-for-SIPI state blocks external interrupts, non-maskable interrupts (NMIs), INIT signals, and system-management interrupts (SMIs). Such events do not cause VM exits if they arrive while a logical processor is in the wait-for-SIPI state and in VMX non-root operation.

29.7.3 Delivery of Pending Debug Exceptions after VM Entry

The pending debug exceptions field in the guest-state area indicates whether there are debug exceptions that have not yet been delivered (see Section •). This section describes how these are treated on VM entry.

There are no pending debug exceptions after VM entry if any of the following are true:

- The VM entry is **injecting using FRED event delivery** or with one of the following **event** types: external interrupt, non-maskable interrupt (NMI), hardware exception, or privileged software exception.
- The interruptibility-state field does not indicate blocking by MOV SS and the VM entry is **injecting** with either of the following **event** types: software interrupt or software exception.
- The VM entry is not **injecting** and the activity-state field indicates either shutdown or wait-for-SIPI.

If none of the above hold, the pending debug exceptions field specifies the debug exceptions that are pending for the guest. There are **valid pending debug exceptions** if either the BS bit (bit 14) or the enable-breakpoint bit (bit 12) is 1. If there are valid pending debug exceptions, they are handled as follows:

- If the VM entry is not **injecting**, the pending debug exceptions are treated as they would had they been encountered normally in guest execution:
 - If the logical processor is not blocking such exceptions (the interruptibility-state field indicates no blocking by MOV SS), a debug exception is delivered after VM entry (see below).
 - If the logical processor is blocking such exceptions (due to blocking by MOV SS), the pending debug exceptions are held pending or lost as would normally be the case.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. See Chapter 7, "Safer Mode Extensions Reference," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D.

- If the VM entry is **injecting** (using **IDT event delivery**, with **event** type software interrupt or software exception, and with blocking by MOV SS), the following items apply:
 - For injection of a software interrupt or of a software exception with vector 3 (#BP) or vector 4 (#OF) — or a privileged software exception with vector 1 (#DB) — the pending debug exceptions are treated as they would had they been encountered normally in guest execution if the corresponding instruction (INT1, INT3, or INTO) were executed after a MOV SS that encountered a debug trap.
 - For injection of a software exception with a vector other than 3 and 4, the pending debug exceptions may be lost or they may be delivered after injection (see below).

If there are no valid pending debug exceptions (as defined above), no pending debug exceptions are delivered after VM entry.

If a pending debug exception is delivered after VM entry, it has the priority of “traps on the previous instruction” (see Section 7.9 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). Thus, INIT signals and system-management interrupts (SMIs) take priority of such an exception, as do VM exits induced by the TPR threshold (see Section 29.7.7) and pending MTF VM exits (see Section 29.7.8). The exception takes priority over any pending non-maskable interrupt (NMI) or external interrupt and also over VM exits due to the 1-settings of the “interrupt-window exiting” and “NMI-window exiting” VM-execution controls.

FRED event delivery saves as part of event information a bit that indicates whether event occurred while the logical processor was in enclave mode. For a pending debug exception is delivered after VM entry, the value saved is that of bit 4 of the guest interruptibility-state field (which indicates “enclave interruption”).

A pending debug exception delivered after VM entry causes a VM exit if the bit 1 (#DB) is 1 in the exception bitmap. If it does not cause a VM exit, it updates DR6 normally.

29.7.4 VMX-Preemption Timer

If the “activate VMX-preemption timer” VM-execution control is 1, VM entry starts the VMX-preemption timer with the unsigned value in the VMX-preemption timer-value field.

It is possible for the VMX-preemption timer to expire during VM entry (e.g., if the value in the VMX-preemption timer-value field is zero). If this happens (and if the VM entry was not to the wait-for-SIPI state), a VM exit occurs with its normal priority after any event injection and before execution of any instruction following VM entry. For example, any pending debug exceptions established by VM entry (see Section 29.7.3) take priority over a timer-induced VM exit. (The timer-induced VM exit will occur after delivery of the debug exception, unless that exception or its delivery causes a different VM exit.)

See Section 28.5.1 for details of the operation of the VMX-preemption timer in VMX non-root operation, including the blocking and priority of the VM exits that it causes.

29.7.5 Interrupt-Window Exiting and Virtual-Interrupt Delivery

If “interrupt-window exiting” VM-execution control is 1, an open interrupt window may cause a VM exit immediately after VM entry (see Section 28.2 for details). If the “interrupt-window exiting” VM-execution control is 0 but the “virtual-interrupt delivery” VM-execution control is 1, a virtual interrupt may be delivered immediately after VM entry (see Section 29.3.2.5 and Section 32.2.1).

The following items detail the treatment of these events:

- These events occur after any event injection specified for VM entry.
- Non-maskable interrupts (NMIs) and higher priority events take priority over these events. These events take priority over external interrupts and lower priority events.
- These events wake the logical processor if it just entered the HLT state because of a VM entry (see Section 29.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

29.7.6 NMI-Window Exiting

The “NMI-window exiting” VM-execution control may cause a VM exit to occur immediately after VM entry (see Section 28.2 for details).

The following items detail the treatment of these VM exits:

- These VM exits follow event injection if such injection is specified for VM entry.
- Debug-trap exceptions (see Section 29.7.3) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.
- VM exits caused by this control wake the logical processor if it just entered either the HLT state or the shutdown state because of a VM entry (see Section 29.7.2). They do not occur if the logical processor just entered the wait-for-SIPI state.

29.7.7 VM Exits Induced by the TPR Threshold

If the “use TPR shadow” and “virtualize APIC accesses” VM-execution controls are both 1 and the “virtual-interrupt delivery” VM-execution control is 0, a VM exit occurs immediately after VM entry if the value of bits 3:0 of the TPR threshold VM-execution control field is greater than the value of bits 7:4 of VTPR (see Section 32.1.1).¹

The following items detail the treatment of these VM exits:

- The VM exits are not blocked if RFLAGS.IF = 0 or by the setting of bits in the interruptibility-state field in guest-state area.
 - The VM exits follow event injection if such injection is specified for VM entry.
 - VM exits caused by this control take priority over system-management interrupts (SMIs), INIT signals, and lower priority events. They thus have priority over the VM exits described in Section 29.7.5, Section 29.7.6, and Section 29.7.8, as well as any interrupts or debug exceptions that may be pending at the time of VM entry.
 - These VM exits wake the logical processor if it just entered the HLT state as part of a VM entry (see Section 29.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.
- If such a VM exit is suppressed because the processor just entered the shutdown state, it occurs after the delivery of any event that cause the logical processor to leave the shutdown state while remaining in VMX non-root operation (e.g., due to an NMI that occurs while the “NMI-exiting” VM-execution control is 0).
- The basic exit reason is “TPR below threshold.”

29.7.8 Pending MTF VM Exits

As noted in Section 29.6.2, VM entry may cause an MTF VM exit to be pending immediately after VM entry. The following items detail the treatment of these VM exits:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these VM exits. These VM exits take priority over debug-trap exceptions and lower priority events.
- These VM exits wake the logical processor if it just entered the HLT state because of a VM entry (see Section 29.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

29.7.9 VM Entries and Advanced Debugging Features

VM entries are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

1. “Virtualize APIC accesses” and “virtual-interrupt delivery” are secondary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if these controls were 0. See Section 27.6.2.

29.7.10 User-Interrupt Recognition After VM Entry

A VM entry results in recognition of a pending user interrupt if it completes with $CR4.UINTR = IA32\_EFER.LMA = 1$ and with $UIRR \neq 0$; otherwise, no pending user interrupt is recognized.

29.8 VM-ENTRY FAILURES DURING OR AFTER LOADING GUEST STATE

VM-entry failures due to the checks identified in Section 29.3.1 and failures during the MSR loading identified in Section 29.4 are treated differently from those that occur earlier in VM entry. In these cases, the following steps take place:

1. Information about the VM-entry failure is recorded in the VM-exit information fields:
 - Exit reason.
 - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM-entry failure. The following numbers are used:
 33. VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 29.3.1.
 34. VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs (see Section 29.4).
 41. VM-entry failure due to machine-check event. A machine-check event occurred during VM entry (see Section 29.9).
 - Bit 31 is set to 1 to indicate a VM-entry failure.
 - The remainder of the field (bits 30:16) is cleared.
 - Exit qualification. This field is set based on the exit reason.
 - VM-entry failure due to invalid guest state. In most cases, the exit qualification is cleared to 0. The following non-zero values are used in the cases indicated:
 1. Not used.
 2. Failure was due to a problem loading the PDPTes (see Section 29.3.1.6).
 3. Failure was due to an attempt to inject a non-maskable interrupt (NMI) into a guest that is blocking events through the STI blocking bit in the interruptibility-state field.
 4. Failure was due to an invalid VMCS link pointer (see Section 29.3.1.5).

VM-entry checks on guest-state fields may be performed in any order. Thus, an indication by exit qualification of one cause does not imply that there are not also other errors. Different processors may give different exit qualifications for the same VMCS.
 - VM-entry failure due to MSR loading. The exit qualification is loaded to indicate which entry in the VM-entry MSR-load area caused the problem (1 for the first entry, 2 for the second, etc.).
 - All other VM-exit information fields are unmodified.
2. Processor state is loaded as would be done on a VM exit (see Section 30.5). If this results in $[CR4.PAE \& CR0.PG \& \sim IA32\_EFER.LMA] = 1$, page-directory-pointer-table entries (PDPTes) may be checked and loaded (see Section 30.5.4).
3. The state of blocking by NMI is what it was before VM entry.
4. MSRs are loaded as specified in the VM-exit MSR-load area (see Section 30.6).

Although this process resembles that of a VM exit, many steps taken during a VM exit do not occur for these VM-entry failures:

- Most VM-exit information fields are not updated (see step 1 above).
- The valid bit in the [injected-event identification](#) field is not cleared.
- The guest-state area is not modified.
- No MSRs are saved into the VM-exit MSR-store area.

29.9 MACHINE-CHECK EVENTS DURING VM ENTRY

If a machine-check event occurs during a VM entry, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM entry:
 - If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:¹
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine-check condition.”
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If CR4.MCE = 1, a machine-check exception (#MC) is delivered **normally**.
- The machine-check event is handled after VM entry completes:
 - If the VM entry ends with CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If the VM entry ends with CR4.MCE = 1, a machine-check exception (#MC) is generated:
 - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered **normally after VM entry**.
 - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- A VM-entry failure occurs as described in Section 29.8. The basic exit reason is 41, for “VM-entry failure due to machine-check event.”

The first option is not used if the machine-check event occurs after any guest state has been loaded. The second option is used only if VM entry is able to load all guest state.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 7, “Safer Mode Extensions Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

24. Updates to Chapter 30, Volume 3C

Change bars and text show changes to Chapter 30 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- Updated Section 30.1, "Architectural State Before a VM Exit" for FRED by removing reference to IDT and replacing VM-exit with exiting-event.
- In Section 30.2, replaced VM-entry with injected-event.
- In Section 30.2.1, replaced IDT-vectoring information with original-event identification.
- In Section 30.2.3, replaced VM-exit with exiting-event, updated bit 11, and added bit 13 (nested exception) for FRED.
- Replaced IDT-vectoring information with original event identification and VM-exit with exiting-event in Section 30.2.3, "Information About NMI Unblocking Due to IRET".
- Replaced IDT-vectoring information with original event identification, added a note for SYSCALL and SYSENTER instructions when used with FRED, documented bit 13 (nested exception), and added original-event data in Section 30.2.4, "Information for VM Exits During Event Delivery".
- Updated case when VM exits due to faults encountered during event delivery with FRED in Section 30.2.5, "Information for VM Exits Due to Instruction Execution" and updated IDT-vectoring with original-event.
- Added "save FRED" VM-exit control in Section 30.3.1, "Saving Control Registers, Debug Registers, and MSRs".
- Removed reference to IDT in Section 30.3.3, "Saving RIP, RSP, RFLAGS, and SSP" and replaced IDT-vectoring with original-event.
- In Section 30.5.1, "Loading Host Control Registers, Debug Registers, MSRs", added CR4.FRED bit and "load FRED" VM-exit control bit setting for MSRs.
- Removed reference to IDT in Section 30.8, "Machine-Check Events During VM Exit".

VM exits occur in response to certain instructions and events in VMX non-root operation as detailed in Section 28.1 through Section 28.2. VM exits perform the following operations:

1. Information about the cause of the VM exit is recorded in the VM-exit information fields and VM-entry control fields are modified as described in Section 30.2.
2. Processor state is saved in the guest-state area (Section 30.3).
3. MSRs may be saved in the VM-exit MSR-store area (Section 30.4). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.
4. The following may be performed in parallel and in any order (Section 30.5):
 - Processor state is loaded based in part on the host-state area and some VM-exit controls. This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM. See Section 34.15.6 for information on how processor state is loaded by such VM exits.
 - Address-range monitoring is cleared.
5. MSRs may be loaded from the VM-exit MSR-load area (Section 30.6). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.

VM exits are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

Section 30.1 clarifies the nature of the architectural state before a VM exit begins. The steps described above are detailed in Section 30.2 through Section 30.6.

Section 34.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, ordinary transitions to SMM are replaced by VM exits to a separate SMM monitor. Called **SMM VM exits**, these are caused by the arrival of an SMI or the execution of VMCALL in VMX root operation. SMM VM exits differ from other VM exits in ways that are detailed in Section 34.15.2.

30.1 ARCHITECTURAL STATE BEFORE A VM EXIT

This section describes the architectural state that exists before a VM exit, especially for VM exits caused by events that would normally be delivered [to guest software](#). Note the following:

- An exception causes a VM exit **directly** if the bit corresponding to that exception is set in the exception bitmap. A non-maskable interrupt (NMI) causes a VM exit directly if the “NMI exiting” VM-execution control is 1. An external interrupt causes a VM exit directly if the “external-interrupt exiting” VM-execution control is 1. A start-up IPI (SIPI) that arrives while a logical processor is in the wait-for-SIPI activity state causes a VM exit directly. INIT signals that arrive while the processor is not in the wait-for-SIPI activity state cause VM exits directly.
- An exception, NMI, external interrupt, or software interrupt causes a VM exit **indirectly** if it does not do so directly but delivery of the event causes a nested exception, double fault, task switch, APIC access (see Section 32.4), EPT violation, EPT misconfiguration, page-modification log-full event (see Section 31.3.6), or SPP-related event (see Section 31.3.4) that causes a VM exit.
- An event **results** in a VM exit if it causes a VM exit (directly or indirectly).

The following bullets detail when architectural state is and is not updated in response to VM exits:

- If an event causes a VM exit directly, it does not update architectural state as it would have if it had it not caused the VM exit:
 - A debug exception does not update DR6, DR7, or IA32_DEBUGCTL. (Information about the nature of the debug exception is saved in the exit qualification field.)
 - A page fault does not update CR2. (The linear address causing the page fault is saved in the exit-qualification field.)

- An NMI causes subsequent NMIs to be blocked, but only after the VM exit completes.
- An external interrupt does not acknowledge the interrupt controller and the interrupt remains pending, unless the “acknowledge interrupt on exit” VM-exit control is 1. In such a case, the interrupt controller is acknowledged and the interrupt is no longer pending.
- The flags L0 – L3 in DR7 (bit 0, bit 2, bit 4, and bit 6) are not cleared when a task switch causes a VM exit.
- If a task switch causes a VM exit, none of the following are modified by the task switch: old task-state segment (TSS); new TSS; old TSS descriptor; new TSS descriptor; RFLAGS.NT¹; or the TR register.
- No last-exception record is made if the event that would do so directly causes a VM exit.
- If a machine-check exception causes a VM exit directly, this does not prevent machine-check MSRs from being updated. These are updated by the machine-check event itself and not the resulting machine-check exception.
- If the logical processor is in an inactive state (see Section •) and not executing instructions, some events may be blocked but others may return the logical processor to the active state. Unblocked events may cause VM exits.² If an unblocked event causes a VM exit directly, a return to the active state occurs only after the VM exit completes.³ The VM exit generates any special bus cycle that is normally generated when the active state is entered from that activity state.

MTF VM exits (see Section 28.5.2 and Section 29.7.8) are not blocked in the HLT activity state. If an MTF VM exit occurs in the HLT activity state, the logical processor returns to the active state only after the VM exit completes. MTF VM exits are blocked the shutdown state and the wait-for-SIPI state.

- If an event causes a VM exit indirectly, the event does update architectural state:
 - A debug exception updates DR6, DR7, and the IA32_DEBUGCTL MSR. No debug exceptions are considered pending.
 - A page fault updates CR2.
 - An NMI causes subsequent NMIs to be blocked before the VM exit commences.
 - An external interrupt acknowledges the interrupt controller and the interrupt is no longer pending.
 - If the logical processor had been in an inactive state, it enters the active state and, before the VM exit commences, generates any special bus cycle that is normally generated when the active state is entered from that activity state.
 - There is no blocking by STI or by MOV SS when the VM exit commences.
 - Processor state that is normally updated as part of delivery through the IDT (CS, RIP, SS, RSP, RFLAGS) is not modified. However, the incomplete delivery of the event may write to the stack.
 - The treatment of last-exception records is implementation dependent:
 - Some processors make a last-exception record when beginning the delivery of an event (before it can encounter a nested exception). Such processors perform this update even if the event encounters a nested exception that causes a VM exit (including the case where nested exceptions lead to a triple fault).
 - Other processors delay making a last-exception record until event delivery has reached some event handler successfully (perhaps after one or more nested exceptions). Such processors do not update the last-exception record if a VM exit or triple fault occurs before an event handler is reached.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

2. If a VM exit takes the processor from an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

3. An exception is made if the logical processor had been inactive due to execution of MWAIT; in this case, it is considered to have become active before the VM exit.

- If the “virtual NMIs” VM-execution control is 1, VM entry injects an NMI, and delivery of the NMI causes a nested exception, double fault, task switch, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event, or APIC access that causes a VM exit, virtual-NMI blocking is in effect before the VM exit commences.
- If a VM exit results from a fault, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered during execution of IRET and the “NMI exiting” VM-execution control is 0, any blocking by NMI is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the exit qualification or in the [exiting-event identification](#) field; see Section 30.2.3.
- If a VM exit results from a fault, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered during execution of IRET and the “virtual NMIs” VM-execution control is 1, virtual-NMI blocking is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the exit qualification or in the [exiting-event identification](#) field; see Section 30.2.3.
- Suppose that a VM exit is caused directly by an x87 FPU Floating-Point Error (#MF) or by any of the following events if the event was unblocked due to (and given priority over) an x87 FPU Floating-Point Error: an INIT signal, an external interrupt, an NMI, an SMI; or a machine-check exception. In these cases, there is no blocking by STI or by MOV SS when the VM exit commences.
- Normally, a last-branch record may be made when an event is delivered. However, if such an event results in a VM exit before delivery is complete, no last-branch record is made.
- If machine-check exception results in a VM exit, processor state is suspect and may result in suspect state being saved to the guest-state area. A VM monitor should consult the RIPV and EIPV bits in the IA32_MC_G_STATUS MSR before resuming a guest that caused a VM exit resulting from a machine-check exception.
- If a VM exit results from a fault, APIC access (see Section 32.4), EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered while executing an instruction, data breakpoints due to that instruction may have been recognized and information about them may be saved in the pending debug exceptions field (unless the VM exit clears that field; see Section 30.3.4).
- The following VM exits are considered to happen after an instruction is executed:
 - VM exits resulting from debug traps (single-step, I/O breakpoints, and data breakpoints).
 - VM exits resulting from debug exceptions (data breakpoints) whose recognition was delayed by blocking by MOV SS.
 - VM exits resulting from some machine-check exceptions.
 - Trap-like VM exits due to execution of MOV to CR8 when the “CR8-load exiting” VM-execution control is 0 and the “use TPR shadow” VM-execution control is 1 (see Section 32.3). (Such VM exits can occur only from 64-bit mode and thus only on processors that support Intel 64 architecture.)
 - Trap-like VM exits due to execution of WRMSR when the “use MSR bitmaps” VM-execution control is 1; the value of ECX is in the range 800H–8FFH; and the bit corresponding to the ECX value in write bitmap for low MSRs is 0; and the “virtualize x2APIC mode” VM-execution control is 1. See Section 32.5.
 - VM exits caused by APIC-write emulation (see Section 32.4.3.2) that result from APIC accesses as part of instruction execution.

For these VM exits, the instruction’s modifications to architectural state complete before the VM exit occurs. Such modifications include those to the logical processor’s interruptibility state (see Table 27-3). If there had been blocking by MOV SS, POP SS, or STI before the instruction executed, such blocking is no longer in effect.

A VM exit that occurs in enclave mode sets bit 27 of the exit-reason field and bit 4 of the guest interruptibility-state field. Before such a VM exit is delivered, an Asynchronous Enclave Exit (AEX) occurs (see Chapter 40, “Enclave Exiting Events”). An AEX modifies architectural state (Section 40.3). In particular, the processor establishes the following architectural state as indicated:

- The following bits in RFLAGS are cleared: CF, PF, AF, ZF, SF, OF, and RF.
- FS and GS are restored to the values they had prior to the most recent enclave entry.
- RIP is loaded with the AEP of interrupted enclave thread.
- RSP is loaded from the URSP field in the enclave’s state-save area (SSA).

30.2 RECORDING VM-EXIT INFORMATION AND UPDATING VM-ENTRY CONTROL FIELDS

VM exits begin by recording information about the nature of and reason for the VM exit in the VM-exit information fields. Section 30.2.1 to Section 30.2.5 detail the use of these fields.

In addition to updating the VM-exit information fields, the valid bit (bit 31) is cleared in the [injected-event identification](#) field. If bit 5 of the IA32_VMX_MISC MSR (index 485H) is read as 1 (see Appendix A.6), the value of IA32_EFER.LMA is stored into the “IA-32e mode guest” VM-entry control.¹

30.2.1 Basic VM-Exit Information

Section 27.9.1 defines the basic VM-exit information fields. The following items detail their use.

- **Exit reason.**
 - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM exit. Appendix C lists the numbers used and their meaning.

Future processors may introduce new VM exits with new basic exit reasons. Any such VM exit will occur only with VM-execution control settings introduced by the same processor or due to execution of an instruction that would cause an invalid-opcode exception (#UD) on previous processors. Software written for a particular processor generation should be aware that, if run on a later processor generation, it may experience VM exits with new basic exit reasons; it should also be aware that such a VM exit would result only from execution of an instruction that would have caused #UD on the earlier processor generation.
 - Bit 25 is set if the “prematurely busy shadow stack” VM-exit control is 1 and the VM exit caused a shadow stack become prematurely busy (see Section 28.4.3). Otherwise, the bit is cleared.
 - Bit 26 of this field is set to 1 if the VM exit occurred after assertion of a bus lock while the “VMM bus-lock detection” VM-execution control was 1. Such VM exits include those that occur due to the 1-setting of that control as well as others that might occur during execution of an instruction that asserted a bus lock.
 - Bit 27 of this field is set to 1 if the VM exit occurred while the logical processor was in enclave mode.

Such VM exits include those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode.

A VM exit also sets this bit if it was due to or incident to delivery of an event that was either (1) made pending or injected by VM entry while the guest interruptibility-state field indicates an enclave interruption (bit 4 of the field was 1); or (2) pending following an execution of RSM that occurred while bit 1 was set in the byte at offset 7EE0H in the state save image in SMRAM.
 - The remainder of the field (bits 31:28 and bits 24:16) is cleared to 0 (certain SMM VM exits may set some of these bits; see Section 34.15.2.3).²
- **Exit qualification.** This field is saved for VM exits due to the following causes: debug exceptions; page-fault exceptions; start-up IPIs (SIPs); system-management interrupts (SMIs) that arrive immediately after the execution of I/O instructions; task switches; INVEPT; INVLPG; INVPCID; INVVPID; LGDT; LIDT; LLDT; LTR; RDMSRLIST; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; WBINVD; WBNOINVD; WRMSR; WRMSRLIST; WRMSRNS; XRSTORS; XSAVES; control-register accesses; MOV DR; I/O instructions; MWAIT; accesses to the APIC-access page (see Section 32.4); EPT violations (see Section 31.3.3.2); EOI virtualization (see Section 32.1.4); APIC-write emulation (see Section 32.4.3.3); page-modification log full (see Section 31.3.6); SPP-related events (see Section 31.3.4); and instruction timeout (see Section 28.2). For all other VM exits, this field is cleared. The following items provide details:

1. Bit 5 of the IA32_VMX_MISC MSR is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control.

2. Bit 31 of this field is set on certain VM-entry failures; see Section 29.8.

- For a debug exception, the exit qualification contains information about the debug exception. The information has the format given in Table 30-1.

Table 30-1. Exit Qualification for Debug Exceptions

Bit Position(s)	Contents
3:0	B3 – B0. When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if its corresponding enabling bit in DR7 is not set.
10:4	Not currently defined.
11	BLD. When set, this bit indicates that a bus lock was asserted while OS bus-lock detection was enabled and CPL > 0 (see Section 20.3.1.6 (“OS Bus-Lock Detection”)). ¹
12	Not currently defined.
13	BD. When set, this bit indicates that the cause of the debug exception is “debug register access detected.”
14	BS. When set, this bit indicates that the cause of the debug exception is either the execution of a single instruction (if RFLAGS.TF = 1 and IA32_DEBUGCTL.BTF = 0) or a taken branch (if RFLAGS.TF = DEBUGCTL.BTF = 1).
15	Not currently defined.
16	RTM. When set, this bit indicates that a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 17.3.7, “RTM-Enabled Debugger Support,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). ²
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 11 to indicate detection of a bus lock, while this field **sets** the bit to indicate that condition.
2. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 16 to indicate an RTM-related exception, while this field **sets** the bit to indicate that condition.

- For a page-fault exception, the exit qualification contains the linear address that caused the page fault. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata. On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

If the page-fault exception occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the exit qualification are cleared.

- For a start-up IPI (SIPI), the exit qualification contains the SIPI vector information in bits 7:0. Bits 63:8 of the exit qualification are cleared to 0.
- For a task switch, the exit qualification contains details about the task switch, encoded as shown in Table 30-2.
- For INVLPG, the exit qualification contains the linear-address operand of the instruction.
 - On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
 - If the INVLPG source operand specifies an unusable segment, the linear address specified in the exit qualification will match the linear address that the INVLPG would have used if no VM exit occurred. This address is not architecturally defined and may be implementation-specific.
- For INVEPT, INVPCID, INVVPID, LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXON, XRSTORS, and XSAVES, the exit qualification receives the value of the instruction’s displacement field, which is sign-extended to 64 bits if necessary (32 bits on processors

Table 30-2. Exit Qualification for Task Switches

Bit Position(s)	Contents
15:0	Selector of task-state segment (TSS) to which the guest attempted to switch
29:16	Not currently defined
31:30	Source of task switch initiation: 0: CALL instruction 1: IRET instruction 2: JMP instruction 3: Task gate in IDT
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

that do not support Intel 64 architecture). If the instruction has no displacement (for example, has a register operand), zero is stored into the exit qualification.

On processors that support Intel 64 architecture, an exception is made for RIP-relative addressing (used only in 64-bit mode). Such addressing causes an instruction to use an address that is the sum of the displacement field and the value of RIP that references the following instruction. In this case, the exit qualification is loaded with the sum of the displacement field and the appropriate RIP value.

In all cases, bits of this field beyond the instruction's address size are undefined. For example, suppose that the address-size field in the VM-exit instruction-information field (see Section 27.9.4 and Section 30.2.5) reports an n -bit address size. Then bits 63: n (bits 31: n on processors that do not support Intel 64 architecture) of the instruction displacement are undefined.

- For a control-register access, the exit qualification contains information about the access and has the format given in Table 30-3.
- For MOV DR, the exit qualification contains information about the instruction and has the format given in Table 30-4.
- For an I/O instruction, the exit qualification contains information about the instruction and has the format given in Table 30-5.
- For MWAIT, the exit qualification contains a value that indicates whether address-range monitoring hardware was armed. The exit qualification is set either to 0 (if address-range monitoring hardware is not armed) or to 1 (if address-range monitoring hardware is armed).
- For RDMSRLIST and WRMSRLIST, the exit qualification depends on the setting of the “use MSR bitmaps” VM-execution control. If the control is 0, the exit qualification is zero. If the control is 1, the exit qualification is the index of the MSR whose access caused the VM exit (see Section 28.1.3).
- WBINVD and WBNOINVD use the same basic exit reason (see Appendix C). For WBINVD, the exit qualification is 0, while for WBNOINVD it is 1.
- WRMSR and WRMSRNS use the same basic exit reason (see Appendix C). For WRMSR, the exit qualification is 0, while for WRMSRNS it is 1.
- For an APIC-access VM exit resulting from a linear access or a guest-physical access to the APIC-access page (see Section 32.4), the exit qualification contains information about the access and has the format given in Table 30-6.¹

If the access to the APIC-access page occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the exit qualification are cleared.

Such a VM exit that set bits 15:12 of the exit qualification to 0000b (data read during instruction execution) or 0001b (data write during instruction execution) set bit 12—which distinguishes data read from data

1. The exit qualification is undefined if the access was part of the logging of a branch record or a processor-event-based-sampling (PEBS) record to the DS save area. It is recommended that software configure the paging structures so that no address in the DS save area translates to an address on the APIC-access page.

write—to that which would have been stored in bit 1—W/R—of the page-fault error code had the access caused a page fault instead of an APIC-access VM exit. This implies the following:

- For an APIC-access VM exit caused by the CLFLUSH and CLFLUSHOPT instructions, the access type is “data read during instruction execution.”
- For an APIC-access VM exit caused by the ENTER instruction, the access type is “data write during instruction execution.”

Table 30-3. Exit Qualification for Control-Register Accesses

Bit Positions	Contents
3:0	Number of control register (0 for CLTS and LMSW). Bit 3 is always 0 on processors that do not support Intel 64 architecture as they do not support CR8.
5:4	Access type: 0 = MOV to CR 1 = MOV from CR 2 = CLTS 3 = LMSW
6	LMSW operand type: 0 = register 1 = memory For CLTS and MOV CR, cleared to 0
7	Not currently defined
11:8	For MOV CR, the general-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) For CLTS and LMSW, cleared to 0
15:12	Not currently defined
31:16	For LMSW, the LMSW source data For CLTS and MOV CR, cleared to 0
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

- For an APIC-access VM exit caused by the MASKMOVQ instruction or the MASKMOVDQU instruction, the access type is “data write during instruction execution.”
- For an APIC-access VM exit caused by the MONITOR instruction, the access type is “data read during instruction execution.”
- For an APIC-access VM exit caused directly by an access to a linear address in the DS save area (BTS or PEBS), the access type is “linear access for monitoring.”
- For an APIC-access VM exit caused by a guest-physical access performed for an access to the DS save area (e.g., to access a paging structure to translate a linear address), the access type is “guest-physical access for monitoring or trace.”

- For an APIC-access VM exit caused by trace-address pre-translation (TAPT) when the “Intel PT uses guest physical addresses” VM-execution control is 1, the access type is “guest-physical access for monitoring or trace.”

Such a VM exit stores 1 for bit 31 in the original-event identification field (see Section 30.2.4) if and only if it sets bits 15:12 of the exit qualification to 0011b (linear access during event delivery) or 1010b (guest-physical access during event delivery).

See Section 32.4.4 for further discussion of these instructions and APIC-access VM exits.

For APIC-access VM exits resulting from physical accesses to the APIC-access page (see Section 32.4.6), the exit qualification is undefined.

- For an EPT violation, the exit qualification contains information about the access causing the EPT violation and has the format given in Table 30-7.

As noted in that table, the format and meaning of the exit qualification depends on the setting of the “mode-based execute control for EPT” VM-execution control and whether the processor supports advanced VM-exit information for EPT violations.¹

An EPT violation that occurs during as a result of execution of a read-modify-write operation sets bit 1 (data write). Whether it also sets bit 0 (data read) is implementation-specific and, for a given implementation, may differ for different kinds of read-modify-write operations.

Table 30-4. Exit Qualification for MOV DR

Bit Position(s)	Contents
2:0	Number of debug register
3	Not currently defined
4	Direction of access (0 = MOV to DR; 1 = MOV from DR)
7:5	Not currently defined
11:8	General-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8 -15 = R8 - R15, respectively
63:12	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

1. Software can determine whether advanced VM-exit information for EPT violations is supported by consulting the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10).

Table 30-5. Exit Qualification for I/O Instructions

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Not currently defined
31:16	Port number (as specified in DX or in an immediate operand)
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

Table 30-6. Exit Qualification for APIC-Access VM Exits from Linear Accesses and Guest-Physical Accesses

Bit Position(s)	Contents
11:0	- If the APIC-access VM exit is due to a linear access, the offset of access within the APIC page. - Undefined if the APIC-access VM exit is due a guest-physical access
15:12	Access type: 0 = linear access for a data read during instruction execution 1 = linear access for a data write during instruction execution 2 = linear access for an instruction fetch 3 = linear access (read or write) during event delivery 4 = linear access for monitoring 10 = guest-physical access during event delivery 11 = guest-physical access for monitoring or trace 15 = guest-physical access for an instruction fetch or during instruction execution Other values not used
16	This bit is set for certain accesses that are asynchronous to instruction execution and not part of event delivery. These includes guest-physical accesses related to trace output by Intel PT (see Section 28.5.4), accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 22.9.5), and accesses that occur during user-interrupt delivery (see Section 9.4.2).
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

Bit 12 reports “NMI unblocking due to IRET”; see Section 30.2.3.

Bit 16 is set for certain accesses that are asynchronous to instruction execution and not part of event delivery. These include trace-address pre-translation (TAPT) for Intel PT (see Section 28.5.4), accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 22.9.5), and accesses as part of user-interrupt delivery (see Section 9.4.2).

- For VM exits caused as part of EOI virtualization (Section 32.1.4), bits 7:0 of the exit qualification are set to vector of the virtual interrupt that was dismissed by the EOI virtualization. Bits above bit 7 are cleared.

- For APIC-write VM exits (Section 32.4.3.3), bits 11:0 of the exit qualification are set to the page offset of the write access that caused the VM exit.¹ Bits above bit 11 are cleared.
- For a VM exit due to a page-modification log-full event (Section 31.3.6), bit 12 of the exit qualification reports “NMI unblocking due to IRET” (see Section 30.2.3). Bit 16 is set if the VM exit occurs during TAPT, EPT-friendly PEBS, or user-interrupt delivery. All other bits of the exit qualification are undefined.
- For a VM exit due to an SPP-related event (Section 31.3.4), bit 11 of the exit qualification indicates the type of event: 0 indicates an SPP misconfiguration and 1 indicates an SPP miss. Bit 12 of the exit qualification reports “NMI unblocking due to IRET” (see Section 30.2.3). Bit 16 is set if the VM exit occurs during TAPT, EPT-friendly PEBS, or user-interrupt delivery. All other bits of the exit qualification are undefined.
- If the “PASID translation” VM-execution control, PASID translation is performed for executions of the ENQCMD and ENQCMLS instructions (see Section 28.5.8). PASID translation may fail, resulting in a VM exit. Such a VM exit saves an exit qualification specified in the following items:
 - For ENQCMD, the exit qualification is IA32_PASID[19:0].
 - For ENQCMLS, the exit qualification contains the low 32 bits of the instruction’s source operand (which had been read from memory prior to PASID translation).
- For a VM exit due to an instruction timeout (Section 28.2), bit 0 indicates (if set) that the context of the virtual machine is invalid and that the VM should not be resumed. Bit 12 of the exit qualification reports “NMI unblocking due to IRET” (see Section 30.2.3). All other bits of the exit qualification are undefined.
- **Guest linear address.** For some VM exits, this field receives a linear address that pertains to the VM exit. The field is set for different VM exits as follows:
 - VM exits due to attempts to execute LMSW with a memory operand. In these cases, this field receives the linear address of that operand. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.
 - VM exits due to attempts to execute INS or OUTS for which the relevant segment is usable (if the relevant segment is not usable, the value is undefined). (ES is always the relevant segment for INS; for OUTS, the relevant segment is DS unless overridden by an instruction prefix.) The linear address is the base address of relevant segment plus (E)DI (for INS) or (E)SI (for OUTS). Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit. If linear-address masking had been in effect (Section 4.4), the address recorded is the original address before any masking (and may thus contain any metadata).
 - VM exits due to EPT violations that set bit 7 of the exit qualification (see Table 30-7; these are all EPT violations except those resulting from an attempt to load the PDPTES as of execution of the MOV CR instruction and those due to TAPT). The linear address may translate to the guest-physical address whose access caused the EPT violation. Alternatively, translation of the linear address may reference a paging-structure entry whose access caused the EPT violation. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.

If the EPT violation occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.
 - VM exits due to SPP-related events. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.
 - If the “prematurely busy shadow stack” VM-exit control is 1, certain VM exits (besides those noted above) save the linear address that pertains to the VM exit if the VM exit caused a shadow stack to become prematurely busy (see Section 28.4.3). This is true for VM exits due for these reasons: EPT misconfiguration, page-modification log-full event, and instruction timeout. (A VM exit due to instruction timeout that sets bit 0 of the exit qualification, indicating that VM context is invalid, does not save a valid linear address.) If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.
 - For all other VM exits, the field is undefined.

1. Execution of WRMSR with ECX = 83FH (self-IPI MSR) can lead to an APIC-write VM exit; the exit qualification for such an APIC-write VM exit is 3F0H.

- **Guest-physical address.** For a VM exit due to an EPT violation, an EPT misconfiguration, or an SPP-related event, this field receives the guest-physical address that caused the EPT violation or EPT misconfiguration. For all other VM exits, the field is undefined.

If the EPT violation or EPT misconfiguration occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.

Table 30-7. Exit Qualification for EPT Violations

Bit Position(s)	Contents
0	Set if the access causing the EPT violation was a data read. ¹
1	Set if the access causing the EPT violation was a data write. ¹
2	Set if the access causing the EPT violation was an instruction fetch.
3	The logical-AND of bit 0 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was readable). ²
4	The logical-AND of bit 1 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was writeable). ²
5	The logical-AND of bit 2 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation. ² If the “mode-based execute control for EPT” VM-execution control is 0, this indicates whether the guest-physical address was executable. If that control is 1, this indicates whether the guest-physical address was executable for supervisor-mode linear addresses.
6	If the “mode-based execute control” VM-execution control is 0, the value of this bit is undefined. If that control is 1, this bit is the logical-AND of bit 10 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation. In this case, it indicates whether the guest-physical address was executable for user-mode linear addresses. ³
7	Set if the guest linear-address field is valid. The guest linear-address field is valid for all EPT violations except those resulting from an attempt to load the guest PDPTes as part of the execution of the MOV CR instruction and those due to trace-address pre-translation (TAPT; Section 28.5.4).
8	If bit 7 is 1: ▪ Set if the access causing the EPT violation is to a guest-physical address that is the translation of a linear address. ▪ Clear if the access causing the EPT violation is to a paging-structure entry as part of a page walk or the update of an accessed or dirty bit. Reserved if bit 7 is 0 (cleared to 0).
9	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ⁴ this bit is 0 if the linear address is a supervisor-mode linear address and 1 if it is a user-mode linear address. (If CR0.PG = 0, the translation of every linear address is a user-mode linear address and thus this bit will be 1.) Otherwise, this bit is undefined.
10	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ⁴ this bit is 0 if paging translates the linear address to a read-only page and 1 if it translates to a read/write page. (If CR0.PG = 0, every linear address is read/write and thus this bit will be 1.) Otherwise, this bit is undefined.
11	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ⁴ this bit is 0 if paging translates the linear address to an executable page and 1 if it translates to an execute-disable page. (If CR0.PG = 0, CR4.PAE = 0, or IA32_EFER.NXE = 0, every linear address is executable and thus this bit will be 0.) Otherwise, this bit is undefined.
12	NMI unblocking due to IRET (see Section 30.2.3).

Table 30-7. Exit Qualification for EPT Violations (Contd.)

Bit Position(s)	Contents
13	Set if the access causing the EPT violation was a shadow-stack access.
14	If supervisor shadow-stack control is enabled (by setting bit 7 of EPTP), this bit is the same as bit 60 in the EPT paging-structure entry that maps the page of the guest-physical address of the access causing the EPT violation. Otherwise (or if translation of the guest-physical address terminates before reaching an EPT paging-structure entry that maps a page), this bit is undefined.
15	This bit is set if the EPT violation was caused as a result of guest-paging verification. See Section 31.3.3.2.
16	This bit is set if the access was asynchronous to instruction execution not the result of event delivery. The bit is set if the access is related to trace output by Intel PT (see Section 28.5.4), accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 22.9.5), or to user-interrupt delivery (see Section 9.4.2). Otherwise, this bit is cleared.
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. If accessed and dirty flags for EPT are enabled, processor accesses to guest paging-structure entries are treated as writes with regard to EPT violations (see Section 31.3.3.2). If such an access causes an EPT violation, the processor sets both bit 0 and bit 1 of the exit qualification.
2. Bits 5:3 are cleared to 0 if either (1) any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present; or (2) 4-level EPT is in use and the guest-physical address sets any bits in the range 51:48 (see Section 31.3.2).
3. Bit 6 is cleared to 0 if (1) the “mode-based execute control” VM-execution control is 1; and (2) either (a) any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present; or (b) 4-level EPT is in use and the guest-physical address sets any bits in the range 51:48 (see Section 31.3.2).
4. Software can determine whether advanced VM-exit information for EPT violations is supported by consulting the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10).

30.2.2 Information for VM Exits Due to Vectored Events

Section 27.9.2 defines fields containing information for VM exits due to the following events: exceptions (including those generated by the instructions INT1, INT3, INTO, BOUND, UD1, UD2, and UDB); external interrupts that occur while the “acknowledge interrupt on exit” VM-exit control is 1; and non-maskable interrupts (NMIs).¹ Such VM exits include those that occur on an attempt at a task switch that causes an exception before generating the VM exit due to the task switch that causes the VM exit.

The following items detail the use of these fields:

- **Exiting-event identification** (format given in Table 27-20). The following items detail how this field is established for VM exits due to these events:
 - For an exception, bits 7:0 receive the exception vector (at most 31). For an NMI, bits 7:0 are set to 2. For an external interrupt, bits 7:0 receive the vector.
 - Bits 10:8 are set to 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 5 (privileged software exception), or 6 (software exception). Hardware exceptions comprise all exceptions except the following:
 - Debug exceptions (#DB) generated by the INT1 instruction; these are privileged software exceptions. (Other debug exceptions are considered hardware exceptions, as are those caused by executions of INT1 in enclave mode.)
 - Breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. (A #BP that occurs in enclave mode is considered a hardware exception.)

1. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with value 1 or 3 for *n*.

BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD0, UD1, UD2, and UDB are hardware exceptions.

- Bit 11 is set to 1 if the VM exit is caused by a hardware exception that would have delivered an error code on the stack; otherwise, it is 0.¹ This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).² If bit 11 is set to 1, the error code is placed in the **exiting-event** error code (see below).
- Bit 12 reports “NMI unblocking due to IRET”; see Section 30.2.3. The value of this bit is undefined if the VM exit is due to a double fault (the **event** type is hardware exception and the vector is 8).
- Bit 13 (nested exception) is set to 1 if the VM exit was caused by a hardware exception that was encountered during FRED event delivery of some other event; it is not set for VM exits due to #DF or to those encountered during IDT event delivery. Other VM exits for which the field is valid (including VM exits due to #DF) save bit 13 as 0.
- Bits 30:14 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits (including those due to external interrupts when the “acknowledge interrupt on exit” VM-exit control is 0), the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- **Exiting-event error code.**

- For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the **exiting-event identification** field, this field receives the error code that would have been pushed on the stack had the event causing the VM exit been delivered normally. The EXT bit is set in this field exactly when it would be set normally. For exceptions that occur during the delivery of double fault (if the **original-event identification** field indicates a double fault), the EXT bit is set to 1, assuming that (1) that the exception would produce an error code normally (if not incident to double-fault delivery) and (2) that the error code uses the EXT bit (not for page faults, which use a different format).
- For other VM exits, the value of this field is undefined.

30.2.3 Information About NMI Unblocking Due to IRET

A VM exit may occur during execution of the IRET instruction for reasons including the following: faults, EPT violations, page-modification log-full events, SPP-related events, or instruction timeouts.

An execution of IRET that commences while non-maskable interrupts (NMIs) are blocked will unblock NMIs even if a fault or VM exit occurs; the state saved by such a VM exit will indicate that NMIs were not blocked.

VM exits for the reasons enumerated above provide more information to software by saving a bit called “NMI unblocking due to IRET.” This bit is defined if (1) either the “NMI exiting” VM-execution control is 0 or the “virtual NMIs” VM-execution control is 1; (2) the VM exit does not set the valid bit in the **original-event identification** field (see Section 30.2.4); and (3) the VM exit is not due to a double fault. In these cases, the bit is defined as follows:

- The bit is 1 if the VM exit resulted from a memory access as part of execution of the IRET instruction and one of the following holds:
 - The “virtual NMIs” VM-execution control is 0 and blocking by NMI (see Table 27-3) was in effect before execution of IRET.
 - The “virtual NMIs” VM-execution control is 1 and virtual-NMI blocking was in effect before execution of IRET.
- The bit is 0 for all other relevant VM exits.

1. FRED event delivery always pushes an error code on the stack (it pushes zero if no error code is defined). Nevertheless, bit 11 here is set only for hardware exceptions that would have pushed an error code with IDT event delivery.

2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, a logical processor cannot be in real-address mode unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

For VM exits due to faults, NMI unblocking due to IRET is saved in bit 12 of the **exiting-event identification** field (Section 30.2.2). For VM exits due to EPT violations, page-modification log-full events, SPP-related events, and instruction timeouts, NMI unblocking due to IRET is saved in bit 12 of the exit qualification (Section 30.2.1).

(Executions of IRET may also incur VM exits due to APIC accesses and EPT misconfigurations. These VM exits do not report information about NMI unblocking due to IRET.)

30.2.4 Information for VM Exits During Event Delivery

Section 27.9.3 defined **original-event fields** containing information for VM exits that occur while delivering an event and as a result of any of the following cases:¹

- An **exception** occurs during event delivery and causes a VM exit (because the bit associated with the **exception** is set to 1 in the exception bitmap).
- A task switch is invoked through a task gate in the IDT. The VM exit occurs due to the task switch only after the initial checks of the task switch pass (see Section 28.4.2).
- Event delivery causes an APIC-access VM exit (see Section 32.4).
- An EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that occurs during event delivery.
- Any of the above VM exits that occur during user-interrupt notification processing (see Section 9.5.2). Such VM exits will be treated as if they occurred during delivery of an external interrupt with the vector UINV.

NOTE

When FRED transitions are enabled, SYSCALL and SYSENTER are delivered using FRED event delivery. Execution of these instructions may result in the VM exits enumerated above (except task switches and user-interrupt notification processing). For that reason, the original-event fields may record information about SYSCALL and SYSENTER when FRED transitions are enabled.

These fields are used for VM exits that occur during delivery of events injected as part of VM entry (see Section 29.6.1.2).

A VM exit is not considered to occur during event delivery in any of the following circumstances:

- The original event causes the VM exit directly (for example, because the original event is a non-maskable interrupt (NMI) and the “NMI exiting” VM-execution control is 1).
- The original event results in a double-fault exception that causes the VM exit directly.
- The VM exit occurred as a result of fetching the first instruction of the handler invoked by the event delivery.
- The VM exit is caused by a triple fault.
- The original event was a software interrupt (INT *n*) executed in virtual-8086 mode with EFLAGS.IOPL < 3 and the VM exit was due to a general-protection exception (#GP) that occurred because either CR4.VME = 0 or bit *n* of the software interrupt redirection bit map in the TSS is set.

The following items detail the use of these fields:

- **Original-event identification** (format given in Table 27-21). The following items detail how this field is established for VM exits that occur during event delivery:
 - If the VM exit occurred during delivery of an exception, bits 7:0 receive the exception vector (at most 31). If the VM exit occurred during delivery of an NMI, bits 7:0 are set to 2. If the VM exit occurred during delivery of an external interrupt, bits 7:0 receive the vector.
 - Bits 10:8 are set to indicate the type of event that was being delivered when the VM exit occurred: 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 4 (software interrupt), 5 (privileged software interrupt), or 6 (software exception).

Hardware exceptions comprise all exceptions except the following:²

1. This includes the case in which a VM exit occurs while delivering a software interrupt (INT *n*) through the 16-bit IVT (interrupt vector table) that is used in virtual-8086 mode with virtual-machine extensions (if RFLAGS.VM = CR4.VME = 1).

- Debug exceptions (#DB) generated by the INT1 instruction; these are privileged software exceptions. (Other debug exceptions are considered hardware exceptions, as are those caused by executions of INT1 in enclave mode.)
- Breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. (A #BP that occurs in enclave mode is considered a hardware exception.)

BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD0, UD1, UD2, and UDB are hardware exceptions.

- Bit 11 is set to 1 if the VM exit occurred during delivery of a hardware exception that would have delivered an error code on the stack; otherwise, it is 0.¹ This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).² If bit 11 is set to 1, the error code is placed in the original-event error code (see below).
- Bit 12 is undefined.
- Bit 13 (nested exception) is set to 1 if the VM exit occurred during delivery of a hardware exception that was encountered during FRED event delivery of some other event; it is not set for VM exits due to #DF or to those encountered during IDT event delivery. Other VM exits for which the original-event identification field is valid (including a VM exits that occurred during delivery of #DF) save bit 13 as 0.
- Bits 30:14 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits, the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- Original-event error code.
 - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the original-event identification field, this field receives the error code that would have been pushed on the stack by the event that was being delivered at the time of the VM exit. The EXT bit is set in this field when it would be set normally.
 - For other VM exits, the value of this field is undefined.
- Original-event data.
 - For VM exits that occur during FRED event delivery, this field receives the event data that would have been pushed on the stack by the event that was being delivered at the time of the VM exit. This is done for all events (if FRED event delivery would have saved zero for event data, that value is saved).
 - For other VM exits (including those that occur when FRED transitions are not enabled), the value of this field is undefined.

30.2.5 Information for VM Exits Due to Instruction Execution

Section 27.9.4 defined fields containing information for VM exits that occur due to instruction execution. (The VM-exit instruction length is also used for VM exits that occur during the delivery of a software interrupt or software exception.) The following items detail their use.

- **VM-exit instruction length.** This field is used in the following cases:
 - For fault-like VM exits due to attempts to execute one of the following instructions that cause VM exits unconditionally (see Section 28.1.2) or based on the settings of VM-execution controls (see Section 28.1.3): CLTS, CPUID, ENCLS, GETSEC, HLT, IN, INS, INVD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLDT, LMSW, LOADIWKEY, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, PCONFIG,

2. In the following items, INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with value 1 or 3 for *n*.

1. FRED event delivery always pushes an error code on the stack (it pushes zero if no error code is defined). Nevertheless, bit 11 here is set only for hardware exceptions that would have pushed an error code with IDT event delivery.
2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, a logical processor cannot be in real-address mode unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

RDMSR, RDPMSR, RDRAND, RDSEED, RDTSC, RDTSCP, RSM, SEAMCALL, SGDT, SIDT, SLDT, STR, TDCALL, SGDT, SIDT, SLDT, STR, TPAUSE, UMWAIT, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, WBINVD, WBNOINVD, WRMSR, XRSTORS, XSETBV, and XSAVES.¹

- For VM exits due to software exceptions (those generated by executions of INT3 or INTO) or privileged software exceptions (those generated by executions of INT1).
- For VM exits due to faults encountered during delivery of a software interrupt, privileged software exception, software exception, or (if FRED transitions are enabled) SYSCALL or SYSENTER.
- For VM exits due to attempts to effect a task switch via instruction execution. These are VM exits that produce an exit reason indicating task switch and either of the following:
 - An exit qualification indicating execution of CALL, IRET, or JMP instruction.
 - An exit qualification indicating a task gate in the IDT and an original-event identification field indicating that the task gate was encountered during delivery of a software interrupt, privileged software exception, or software exception.
- For APIC-access VM exits and for VM exits caused by EPT violations, page-modification log-full events, and SPP-related events encountered during delivery of a software interrupt, privileged software exception, or software exception.²
- For VM exits due to executions of VMFUNC that fail because one of the following is true:
 - EAX indicates a VM function that is not enabled (the bit at position EAX is 0 in the VM-function controls; see Section 28.5.6.2).
 - EAX = 0 and either ECX ≥ 512 or the value of ECX selects an invalid tentative EPTP value (see Section 28.5.6.3).

In all the above cases, this field receives the length in bytes (1–15) of the instruction (including any instruction prefixes) whose execution led to the VM exit (see the next paragraph for one exception).

The cases of VM exits encountered during delivery of a software interrupt, privileged software exception, or software exception include those encountered during delivery of events injected as part of VM entry (see Section 29.6.1.2). If the original event was injected as part of VM entry, this field receives the value of the VM-entry instruction length.

All VM exits other than those listed in the above items leave this field undefined.

If the VM exit occurred in enclave mode, this field is cleared (none of the previous items apply).

Table 30-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS

Bit Position(s)	Content
6:0	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
14:10	Undefined.

1. This item applies only to fault-like VM exits. It does not apply to trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1 or to those following executions of the WRMSR instruction when the “virtualize x2APIC mode” VM-execution control is 1.

2. The VM-exit instruction-length field is not defined following APIC-access VM exits resulting from physical accesses (see Section 32.4.6) even if encountered during delivery of a software interrupt, privileged software exception, software exception, SYSCALL, or SYSENTER.

Table 30-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS (Contd.)

Bit Position(s)	Content
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for VM exits due to execution of INS.
31:18	Undefined.

- VM-exit instruction information.** For VM exits due to attempts to execute INS, INVEPT, INVPCID, INVVPID, LIDT, LGDT, LLDT, LOADIWKEY, LTR, OUTS, RDRAND, RDSEED, SIDT, SGDT, SLDT, STR, TPAUSE, UMWAIT, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXON, XRSTORS, or XSAVES, this field receives information about the instruction that caused the VM exit. The format of the field depends on the identity of the instruction causing the VM exit:
 - For VM exits due to attempts to execute INS or OUTS, the field has the format is given in Table 30-8.¹
 - For VM exits due to attempts to execute INVEPT, INVPCID, or INVVPID, the field has the format is given in Table 30-9.
 - For VM exits due to attempts to execute LIDT, LGDT, SIDT, or SGDT, the field has the format is given in Table 30-10.
 - For VM exits due to attempts to execute LLDT, LTR, SLDT, or STR, the field has the format is given in Table 30-11.
 - For VM exits due to attempts to execute RDRAND or RDSEED, the field has the format is given in Table 30-12.
 - For VM exits due to attempts to execute TPAUSE or UMWAIT, the field has the format is given in Table 30-13.
 - For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, or XSAVES, the field has the format is given in Table 30-14.
 - For VM exits due to attempts to execute VMREAD or VMWRITE, the field has the format is given in Table 30-15.
 - For VM exits due to attempts to execute LOADIWKEY, the field has the format is given in Table 30-16.
For all other VM exits, the field is undefined, unless the VM exit occurred in enclave mode, in which case the field is cleared.
- I/O RCX, I/O RSI, I/O RDI, I/O RIP.** These fields are undefined except for SMM VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions. See Section 34.15.2.3. Note that, if the VM exit occurred in enclave mode, these fields are all cleared.
- MSR data.** An execution of WRMSRLIST may cause a VM exit if it would write to an MSR for which the MSR bitmaps do not allow writes (see Section 28.1.3). Such VM exits save the 64-bit data that would have been written to the MSR into this field in the VMCS.

1. The format of the field was undefined for these VM exits on the first processors to support the virtual-machine extensions. Software can determine whether the format specified in Table 30-8 is used by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

Table 30-9. Format of the VM-Exit Instruction-Information Field as Used for INVEPT, INVPCID, and INVVPID

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for memory instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Reg2 (same encoding as IndexReg above)

Table 30-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).

Table 30-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT (Contd.)

Bit Position(s)	Content
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
11	Operand size: 0: 16-bit 1: 32-bit Undefined for VM exits from 64-bit mode.
14:12	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
29:28	Instruction identity: 0: SGDT 1: SIDT 2: LGDT 3: LIDT
31:30	Undefined.

Table 30-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).

Table 30-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR (Contd.)

Bit Position(s)	Content
29:28	Instruction identity: 0: SLDT 1: STR 2: LLDT 3: LTR
31:30	Undefined.

Table 30-12. Format of the VM-Exit Instruction-Information Field as Used for RDRAND and RDSEED

Bit Position(s)	Content
2:0	Undefined.
6:3	Operand register (destination register): 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture)
10:7	Undefined.
12:11	Operand size: 0: 16-bit 1: 32-bit 2: 64-bit The value 3 is not used.
31:13	Undefined.

Table 30-13. Format of the VM-Exit Instruction-Information Field as Used for TPAUSE and UMWAIT

Bit Position(s)	Content
2:0	Undefined.
6:3	Operand register (source register): 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture)
31:7	Undefined.

Table 30-14. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, and XSAVES

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Undefined.

30.3 SAVING GUEST STATE

VM exits save certain components of processor state into corresponding fields in the guest-state area of the VMCS (see Section 27.4). On processors that support Intel 64 architecture, the full value of each natural-width field (see Section 27.11.2) is saved regardless of the mode of the logical processor before and after the VM exit.

In general, the state saved is that which was in the logical processor at the time the VM exit commences. See Section 30.1 for a discussion of which architectural updates occur at that time.

Table 30-15. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
31:28	Reg2 (same encoding as Reg1 above)

Section 30.3.1 through Section 30.3.4 provide details for how various components of processor state are saved. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

Table 30-16. Format of the VM-Exit Instruction-Information Field as Used for LOADIWKEY

Bit Position(s)	Content
2:0	Undefined.
6:3	Reg1: identifies the first XMM register operand (XMM0–XMM15; values 8–15 are used only on processors that support Intel 64 architecture).
30:7	Undefined.
31:28	Reg2: identifies the second XMM register operand (see above).

30.3.1 Saving Control Registers, Debug Registers, and MSRs

Contents of certain control registers, debug registers, and MSRs are saved as follows:

- The contents of CR0, CR3, CR4, and the IA32_SYSENTER_CS, IA32_SYSENTER_ESP, and IA32_SYSENTER_EIP MSRs are saved into the corresponding fields. Bits 63:32 of the IA32_SYSENTER_CS MSR are not saved. On processors that do not support Intel 64 architecture, bits 63:32 of the IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are not saved.
- If the “save debug controls” VM-exit control is 1, the contents of DR7 and the IA32_DEBUGCTL MSR are saved into the corresponding fields. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always saved data into these fields.
- If the “save IA32_PAT” VM-exit control is 1, the contents of the IA32_PAT MSR are saved into the corresponding field.
- If the “save IA32_EFER” VM-exit control is 1, the contents of the IA32_EFER MSR are saved into the corresponding field.
- If the processor supports either the 1-setting of the “load IA32_BNDCFGS” VM-entry control or that of the “clear IA32_BNDCFGS” VM-exit control, the contents of the IA32_BNDCFGS MSR are saved into the corresponding field.
- If the processor supports either the 1-setting of the “load IA32_RTIT_CTL” VM-entry control or that of the “clear IA32_RTIT_CTL” VM-exit control, the contents of the IA32_RTIT_CTL MSR are saved into the corresponding field.
- If the processor supports the 1-setting of the “load CET” VM-entry control, the contents of the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs are saved into the corresponding fields. On processors that do not support Intel 64 architecture, bits 63:32 of these MSRs are not saved.
- If the processor supports either the 1-setting of the “load guest IA32_LBR_CTL” VM-entry control or that of the “clear IA32_LBR_CTL” VM-exit control, the contents of the IA32_LBR_CTL MSR are saved into the corresponding field.
- If the processor supports the 1-setting of the “load PKRS” VM-entry control, the contents of the IA32_PKRS MSR are saved into the corresponding field.
- If a processor supports user interrupts, every VM exit saves UINV into the guest UINV field in the VMCS (bits 15:8 of the field are cleared).
- If the “save IA32_PERF_GLOBAL_CTL” VM-exit control is 1, the contents of the IA32_PERF_GLOBAL_CTL MSR are saved into the corresponding field.
- If the “save FRED” VM-exit control is 1, the contents of the following MSRs are saved into the corresponding fields: IA32_FRED_CONFIG, IA32_FRED_RSP1, IA32_FRED_RSP2, IA32_FRED_RSP3, IA32_FRED_STKLVL, IA32_FRED_SSP1, IA32_FRED_SSP2, and IA32_FRED_SSP3.
- The value of the SMBASE field is undefined after all VM exits except SMM VM exits. See Section 34.15.2.

30.3.2 Saving Segment Registers and Descriptor-Table Registers

For each segment register (CS, SS, DS, ES, FS, GS, LDTR, or TR), the values saved for the base-address, segment-limit, and access rights are based on whether the register was unusable (see Section 27.4.1) before the VM exit:

- If the register was unusable, the values saved into the following fields are undefined: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in the access-rights field. The following exceptions apply:
 - CS.
 - The base-address and segment-limit fields are saved.
 - The L, D, and G bits are saved in the access-rights field.
 - SS.
 - DPL is saved in the access-rights field.
 - On processors that support Intel 64 architecture, bits 63:32 of the value saved for the base address are always zero.
 - DS and ES. On processors that support Intel 64 architecture, bits 63:32 of the values saved for the base addresses are always zero.
 - FS and GS. The base-address field is saved.
- If the register was not unusable, the values saved into the following fields are those which were in the register before the VM exit: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in access rights.
- Bits 31:17 and 11:8 in the access-rights field are always cleared. Bit 16 is set to 1 if and only if the segment is unusable.

The contents of the GDTR and IDTR registers are saved into the corresponding base-address and limit fields.

30.3.3 Saving RIP, RSP, RFLAGS, and SSP

The contents of the RIP, RSP, RFLAGS, and SSP (shadow-stack pointer) registers are saved as follows:

- The value saved in the RIP field is determined by the nature and cause of the VM exit:
 - If the VM exit occurred in enclave mode, the value saved is the AEP of interrupted enclave thread (the remaining items do not apply).
 - If the VM exit occurs due to by an attempt to execute an instruction that causes VM exits unconditionally or that has been configured to cause a VM exit via the VM-execution controls, the value saved references that instruction.
 - If the VM exit is caused by an occurrence of an INIT signal, a start-up IPI (SIPI), or system-management interrupt (SMI), the value saved is that which was in RIP before the event occurred.
 - If the VM exit occurs due to the 1-setting of either the “interrupt-window exiting” VM-execution control or the “NMI-window exiting” VM-execution control, the value saved is that which would be in the register had the VM exit not occurred.
 - If the VM exit is due to an external interrupt, non-maskable interrupt (NMI), or hardware exception (as defined in Section 30.2.2), the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate,¹ or into the old task-state segment had the event been delivered through a task gate).
 - If the VM exit is due to a triple fault, the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate, or into the old task-state segment had the event been delivered through a task gate) had delivery of the double fault not encountered the nested exception that caused the triple fault.
 - If the VM exit is due to a software exception (due to an execution of INT3 or INTO) or a privileged software exception (due to an execution of INT1), the value saved references the INT3, INTO, or INT1 instruction that caused that exception.
 - Suppose that the VM exit is due to a task switch that was caused by execution of CALL, IRET, or JMP or by execution of a software interrupt (INT *n*), software exception (due to execution of INT3 or INTO), or

1. The reference here is to the full value of RIP before any truncation that would occur had the stack width been only 32 bits or 16 bits.

privileged software exception (due to execution of INT1) that encountered a task gate in the IDT. The value saved references the instruction that caused the task switch (CALL, IRET, JMP, INT *n*, INT3, INTO, INT1).

- Suppose that the VM exit is due to a task switch that was caused by a task gate in the IDT that was encountered for any reason except the direct access by a software interrupt or software exception. The value saved is that which would have been saved in the old task-state segment had the task switch completed normally.
- If the VM exit is due to an execution of MOV to CR8 or WRMSR that reduced the value of bits 7:4 of VTPR (see Section 32.1.1) below that of TPR threshold VM-execution control field (see Section 32.1.2), the value saved references the instruction following the MOV to CR8 or WRMSR.
- If the VM exit was caused by APIC-write emulation (see Section 32.4.3.2) that results from an APIC access as part of instruction execution, the value saved references the instruction following the one whose execution caused the APIC-write emulation.
- The contents of the RSP register are saved into the RSP field.
- With the exception of the resume flag (RF; bit 16), the contents of the RFLAGS register is saved into the RFLAGS field. RFLAGS.RF is saved as follows:
 - If the VM exit occurred in enclave mode, the value saved is 0 (the remaining items do not apply).
 - If the VM exit is caused directly by an event that would normally be delivered **to software**, the value saved is that which would appear in the saved RFLAGS image (either that which would be saved on the stack had the event been delivered through a trap or interrupt gate¹ or into the old task-state segment had the event been delivered through a task gate) had the event been delivered **to guest software**. See below for VM exits due to task switches caused by task gates in the IDT.
 - If the VM exit is caused by a triple fault, the value saved is that which the logical processor would have in RF in the RFLAGS register had the triple fault taken the logical processor to the shutdown state.
 - If the VM exit is caused by a task switch (including one caused by a task gate in the IDT), the value saved is that which would have been saved in the RFLAGS image in the old task-state segment (TSS) had the task switch completed normally without exception.
 - If the VM exit is caused by an attempt to execute an instruction that unconditionally causes VM exits or one that was configured to do with a VM-execution control, the value saved is 0.²
 - For APIC-access VM exits and for VM exits caused by EPT violations, EPT misconfigurations, page-modification log-full events, or SPP-related events, the value saved depends on whether the VM exit occurred during delivery of an event:
 - If the VM exit stored 0 for bit 31 for **original-event identification** field (because the VM exit did not occur during delivery of an event; see Section 30.2.4), the value saved is 1.
 - If the VM exit stored 1 for bit 31 for **original-event identification** field (because the VM exit did occur during delivery of an event), the value saved is the value that would have appeared in the saved RFLAGS image had the event been delivered (see above).
 - For all other VM exits, the value saved is the value RFLAGS.RF had before the VM exit occurred.
- If the processor supports the 1-setting of the “load CET” VM-entry control, the contents of the SSP register are saved into the SSP field.

30.3.4 Saving Non-Register State

Information corresponding to guest non-register state is saved as follows:

-
1. The reference here is to the full value of RFLAGS before any truncation that would occur had the stack width been only 32 bits or 16 bits.
 2. This is true even if RFLAGS.RF was 1 before the instruction was executed. If, in response to such a VM exit, a VM monitor re-enters the guest to re-execute the instruction that caused the VM exit (for example, after clearing the VM-execution control that caused the VM exit), the instruction may encounter a code breakpoint that has already been processed. A VM monitor can avoid this by setting the guest value of RFLAGS.RF to 1 before resuming guest software.

- The activity-state field is saved with the logical processor's activity state before the VM exit.¹ See Section 30.1 for details of how events leading to a VM exit may affect the activity state. If the VM exit occurred during user-interrupt notification processing (see Section 9.5.2) and the logical processor would have entered the HLT state following user-interrupt notification processing, the saved activity state is "HLT".
- The interruptibility-state field is saved to reflect the logical processor's interruptibility before the VM exit.
 - See Section 30.1 for details of how events leading to a VM exit may affect this state.
 - VM exits that end outside system-management mode (SMM) save bit 2 (blocking by SMI) as 0 regardless of the state of such blocking before the VM exit.
 - Bit 3 (blocking by NMI) is treated specially if the "virtual NMIs" VM-execution control is 1. In this case, the value saved for this field does not indicate the blocking of NMIs but rather the state of virtual-NMI blocking.
 - Bit 4 (enclave interruption) is set to 1 if the VM exit occurred while the logical processor was in enclave mode.

Such VM exits includes those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode.

A VM exit also sets this bit if it was due to or incident to delivery of an event that was either (1) made pending or injected by VM entry while the guest interruptibility-state field sets this bit; or (2) pending following an execution of RSM that occurred while bit 1 was set in the byte at offset 7EE0H in the state save image in SMRAM.

- The pending debug exceptions field is saved as clear for all VM exits except the following:
 - A VM exit caused by an INIT signal, a machine-check exception, or a system-management interrupt (SMI).
 - A VM exit with basic exit reason "TPR below threshold",² "virtualized EOI", "APIC write", "monitor trap flag," or "bus-lock detected."
 - A VM exit due to trace-address pre-translation (TAPT; see Section 28.5.4) or due to accesses related to PEBS on processors with the "EPT-friendly" enhancement (see Section 22.9.5). Such VM exits can have basic exit reason "APIC access," "EPT violation," "EPT misconfiguration," "page-modification log full," or "SPP-related event." When due to TAPT or PEBS, these VM exits (with the exception of those due to EPT misconfigurations) set bit 16 of the exit qualification, indicating that they are asynchronous to instruction execution and not part of event delivery.
 - VM exits that are not caused by debug exceptions and that occur while there is MOV-SS blocking of debug exceptions.

For VM exits that do not clear the field, the value saved is determined as follows:

- Each of bits 3:0 may be set if it corresponds to a matched breakpoint. This may be true even if the corresponding breakpoint is not enabled in DR7.
- Suppose that a VM exit is due to an INIT signal, a machine-check exception, or an SMI; or that a VM exit has basic exit reason "TPR below threshold" or "monitor trap flag." In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit.

If the VM exit occurs immediately after VM entry, the value saved may match that which was loaded on VM entry (see Section 29.7.3). Otherwise, the following items apply:

- Bit 12 (enabled breakpoint) is set to 1 in any of the following cases:
 - If there was at least one matched data or I/O breakpoint that was enabled in DR7.
 - If it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 29.7.3) and the VM exit occurred before those exceptions were either delivered or lost.
 - If the XBEGIN instruction was executed immediately before the VM exit and advanced debugging of RTM transactional regions had been enabled (see Section 17.3.7, "RTM-Enabled Debugger

1. If this activity state was an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

2. This item includes VM exits that occur as a result of certain VM entries (Section 29.7.7).

Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). (This does not apply to VM exits with basic exit reason “monitor trap flag.”)

- If a bus lock was asserted while $CPL > 0$ and OS bus-lock detection was enabled.

In other cases, bit 12 is cleared to 0.

- Bit 14 (BS) is set if $RFLAGS.TF = 1$ in either of the following cases:
 - $IA32\_DEBUGCTL.BTF = 0$ and the cause of a pending debug exception was the execution of a single instruction.
 - $IA32\_DEBUGCTL.BTF = 1$ and the cause of a pending debug exception was a taken branch.
- Bit 16 (RTM) is set if a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions had been enabled. (This does not apply to VM exits with basic exit reason “monitor trap flag.”)
- Suppose that a VM exit is due to another reason (but not a debug exception) and occurs while there is MOV-SS blocking of debug exceptions. In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit. If the VM exit occurs immediately after VM entry (no instructions were executed in VMX non-root operation), the value saved may match that which was loaded on VM entry (see Section 29.7.3). Otherwise, the following items apply:
 - Bit 12 (enabled breakpoint) is set to 1 if there was at least one matched data or I/O breakpoint that was enabled in DR7. Bit 12 is also set if it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 29.7.3) and the VM exit occurred before those exceptions were either delivered or lost. In other cases, bit 12 is cleared to 0.
 - The setting of bit 14 (BS) is implementation-specific. However, it is not set if $RFLAGS.TF = 0$ or $IA32\_DEBUGCTL.BTF = 1$.
- The reserved bits in the field are cleared.
- If the “save VMX-preemption timer value” VM-exit control is 1, the value of timer is saved into the VMX-preemption timer-value field. This is the value loaded from this field on VM entry as subsequently decremented (see Section 28.5.1). VM exits due to timer expiration save the value 0. Other VM exits may also save the value 0 if the timer expired during VM exit. (If the “save VMX-preemption timer value” VM-exit control is 0, VM exit does not modify the value of the VMX-preemption timer-value field.)
- If the logical processor supports the 1-setting of the “enable EPT” VM-execution control, values are saved into the four (4) PDPTE fields as follows:
 - If the “enable EPT” VM-execution control is 1 and the logical processor was using PAE paging at the time of the VM exit, the PDPTE values currently in use are saved:¹
 - The values saved into bits 11:9 of each of the fields is undefined.
 - If the value saved into one of the fields has bit 0 (present) clear, the value saved into bits 63:1 of that field is undefined. That value need not correspond to the value that was loaded by VM entry or to any value that might have been loaded in VMX non-root operation.
 - If the value saved into one of the fields has bit 0 (present) set, the value saved into bits 63:12 of the field is a guest-physical address.
 - If the “enable EPT” VM-execution control is 0 or the logical processor was not using PAE paging at the time of the VM exit, the values saved are undefined.

1. A logical processor uses PAE paging if $CRO.PG = 1$, $CR4.PAE = 1$ and $IA32\_EFER.LMA = 0$. See Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM exit functions as if the “enable EPT” VM-execution control were 0. See Section 27.6.2.

30.4 SAVING MSRS

After processor state is saved to the guest-state area, values of MSRs may be stored into the VM-exit MSR-store area (see Section 27.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-store count) is processed in order by storing the value of the MSR indexed by bits 31:0 (as they would be read by RDMSR) into bits 127:64. Processing of an entry fails in either of the following cases:

- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be read only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32_SMBASE is an MSR that can be read only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be saved on VM exits for model-specific reasons. A processor may prevent certain MSRs (based on the value of bits 31:0) from being stored on VM exits, even if they can normally be read by RDMSR. Such model-specific behavior is documented in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.
- Bits 63:32 of the entry are not all 0.
- An attempt to read the MSR indexed by bits 31:0 would cause a general-protection exception if executed via RDMSR with CPL = 0.

A VMX abort occurs if processing fails for any entry. See Section 30.7.

30.5 LOADING HOST STATE

Processor state is updated on VM exits in the following ways:

- Some state is loaded from or otherwise determined by the contents of the host-state area.
- Some state is determined by VM-exit controls.
- Some state is established in the same way on every VM exit.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order.

On processors that support Intel 64 architecture, the full values of each 64-bit field loaded (for example, the base address for GDTR) is loaded regardless of the mode of the logical processor before and after the VM exit.

The loading of host state is detailed in Section 30.5.1 to Section 30.5.5. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

A logical processor is in IA-32e mode after a VM exit only if the “host address-space size” VM-exit control is 1. If the logical processor was in IA-32e mode before the VM exit and this control is 0, a VMX abort occurs. See Section 30.7.

In addition to loading host state, VM exits clear address-range monitoring (Section 30.5.6).

After the state loading described in this section, VM exits may load MSRs from the VM-exit MSR-load area (see Section 30.6). This loading occurs only after the state loading described in this section.

30.5.1 Loading Host Control Registers, Debug Registers, MSRs

VM exits load new values for controls registers, debug registers, and some MSRs:

- CR0, CR3, and CR4 are loaded from the CR0 field, the CR3 field, and the CR4 field, respectively, with the following exceptions:
 - The following bits are not modified:
 - For CR0, ET, CD, NW; bits 63:32 (on processors that support Intel 64 architecture), 28:19, 17, and 15:6; and any bits that are fixed in VMX operation (see Section 26.8).¹
 - For CR3, bits 63:MAXPHYADDR, where MAXPHYADDR is defined as follows:

- Usually, MAXPHYADDR is the processor's supported physical-address width as enumerated in CPUID.80000008H:EAX[7:0] (this width is at most 52).
- If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.¹

This item applies only to processors that support Intel 64 architecture.

- For CR4, any bits that are fixed in VMX operation (see Section 26.8).
- CR4.PAE is set to 1 if the "host address-space size" VM-exit control is 1.
- CR4.PCIDE is set to 0 if the "host address-space size" VM-exit control is 0.
- CR4.FRED is set to 0 if the "host-address-space size" VM-exit control is 0.
- DR7 is set to 400H.
- If the "clear UINV" VM-exit control is 1, VM exit clears UINV.
- The following MSRs are established as follows:
 - The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.
 - The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since that field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
 - The IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_ESP and IA32_SYSENTER_EIP fields, respectively.

If the processor does not support the Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.

If the processor supports the Intel 64 architecture with $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit $N-1$.²

 - The following steps are performed on processors that support Intel 64 architecture:
 - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 30.5.2).
 - The LMA and LME bits in the IA32_EFER MSR are each loaded with the setting of the "host address-space size" VM-exit control.
 - If the "load IA32_PERF_GLOBAL_CTRL" VM-exit control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field. Bits that are reserved in that MSR are maintained with their reserved values.
 - If the "load IA32_PAT" VM-exit control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field. Bits that are reserved in that MSR are maintained with their reserved values.
 - If the "load IA32_EFER" VM-exit control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field. Bits that are reserved in that MSR are maintained with their reserved values.
 - If the "clear IA32_BNDCFGS" VM-exit control is 1, the IA32_BNDCFGS MSR is cleared to 00000000_00000000H; otherwise, it is not modified.
 - If the "clear IA32_RTIT_CTL" VM-exit control is 1, the IA32_RTIT_CTL MSR is cleared to 00000000_00000000H; otherwise, it is not modified.
 - If the "load CET" VM-exit control is 1, the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs are loaded from the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR fields, respectively.

1. Bits 28:19, 17, and 15:6 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. CR0.ET is always 1 and the other bits are always 0.

1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

2. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

If the processor does not support the Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.

If the processor supports the Intel 64 architecture with $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit $N-1$.

- If the “load PKRS” VM-exit control is 1, the IA32_PKRS MSR is loaded from the IA32_PKRS field. Bits 63:32 of that MSR are maintained with zeroes.
- If the “load FRED” VM-exit control is 1, the following MSRs are loaded from the corresponding fields: IA32_FRED_CONFIG, IA32_FRED_RSP1, IA32_FRED_RSP2, IA32_FRED_RSP3, IA32_FRED_STKLVLs, IA32_FRED_SSP1, IA32_FRED_SSP2, and IA32_FRED_SSP3.

With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-exit MSR-load area. See Section 30.6.

30.5.2 Loading Host Segment and Descriptor-Table Registers

Each of the registers CS, SS, DS, ES, FS, GS, and TR is loaded as follows (see below for the treatment of LDTR):

- The selector is loaded from the selector field. The segment is unusable if its selector is loaded with zero. The checks specified in Section 29.2.3 limit the selector values that may be loaded. In particular, CS and TR are never loaded with zero and are thus never unusable. SS can be loaded with zero only on processors that support Intel 64 architecture and only if the VM exit is to 64-bit mode (64-bit mode allows use of segments marked unusable).
- The base address is set as follows:
 - CS. Cleared to zero.
 - SS, DS, and ES. Undefined if the segment is unusable; otherwise, cleared to zero.
 - FS and GS. Undefined (but, on processors that support Intel 64 architecture, canonical) if the segment is unusable and the VM exit is not to 64-bit mode; otherwise, loaded from the base-address field.

If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit $N-1$.¹ The values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.
 - TR. Loaded from the host-state area. If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit $N-1$.
- The segment limit is set as follows:
 - CS. Set to FFFFFFFFH (corresponding to a descriptor limit of FFFFFFFH and a G-bit setting of 1).
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to FFFFFFFFH.
 - TR. Set to 00000067H.
- The type field and S bit are set as follows:
 - CS. Type set to 11 and S set to 1 (execute/read, accessed, non-conforming code segment).
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, type set to 3 and S set to 1 (read/write, accessed, expand-up data segment).
 - TR. Type set to 11 and S set to 0 (busy 32-bit task-state segment).
- The DPL is set as follows:
 - CS, SS, and TR. Set to 0. The current privilege level (CPL) will be 0 after the VM exit completes.
 - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 0.
- The P bit is set as follows:
 - CS, TR. Set to 1.

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

- SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
- On processors that support Intel 64 architecture, CS.L is loaded with the setting of the “host address-space size” VM-exit control. Because the value of this control is also loaded into IA32_EFER.LMA (see Section 30.5.1), no VM exit is ever to compatibility mode (which requires IA32_EFER.LMA = 1 and CS.L = 0).
- D/B.
 - CS. Loaded with the inverse of the setting of the “host address-space size” VM-exit control. For example, if that control is 0, indicating a 32-bit guest, CS.D/B is set to 1.
 - SS. Set to 1.
 - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
 - TR. Set to 0.
- G.
 - CS. Set to 1.
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
 - TR. Set to 0.

The host-state area does not contain a selector field for LDTR. LDTR is established as follows on all VM exits: the selector is cleared to 0000H, the segment is marked unusable and is otherwise undefined.

The base addresses for GDTR and IDTR are loaded from the GDTR base-address field and the IDTR base-address field, respectively. If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N of each base address is set to the value of bit N-1 of that base address. The GDTR and IDTR limits are each set to FFFFH.

30.5.3 Loading Host RIP, RSP, RFLAGS, and SSP

RIP and RSP are loaded from the RIP field and the RSP field, respectively. RFLAGS is cleared, except bit 1, which is always set.

If the “load CET” VM-exit control is 1, SSP (shadow-stack pointer) is loaded from the SSP field.

30.5.4 Checking and Loading Host Page-Directory-Pointer-Table Entries

If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LMA = 0, the logical processor uses **PAE paging**. See Section 5.4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.¹ When in PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTes). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTes and, if they are valid, loads them into the processor (into internal, non-architectural registers).

A VM exit is to a VMM that uses PAE paging if (1) bit 5 (corresponding to CR4.PAE) is set in the CR4 field in the host-state area of the VMCS; and (2) the “host address-space size” VM-exit control is 0. Such a VM exit may check the validity of the PDPTes referenced by the CR3 field in the host-state area of the VMCS. Such a VM exit must check their validity if either (1) PAE paging was not in use before the VM exit; or (2) the value of CR3 is changing as a result of the VM exit. A VM exit to a VMM that does not use PAE paging must not check the validity of the PDPTes.

A VM exit that checks the validity of the PDPTes uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use. If MOV to CR3 would cause a general-protection exception due to the PDPTes that would be loaded (e.g., because a reserved bit is set), a VMX abort occurs (see Section 30.7). If a VM exit to a VMM that uses PAE does not cause a VMX abort, the PDPTes are loaded into the processor as would MOV to CR3, using the value of CR3 being load by the VM exit.

1. On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

30.5.5 Updating Non-Register State

VM exits affect the non-register state of a logical processor as follows:

- A logical processor is always in the active state after a VM exit.
- Event blocking is affected as follows:
 - There is no blocking by STI or by MOV SS after a VM exit.
 - VM exits caused directly by non-maskable interrupts (NMIs) cause blocking by NMI (see Table 27-3). Other VM exits do not affect blocking by NMI. (See Section 30.1 for the case in which an NMI causes a VM exit indirectly.)
- There are no pending debug exceptions after a VM exit.

Section 31.4 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM exits invalidate cached mappings:

- If the “enable VPID” VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EPTRTA values (EPTRTA is the value of bits 51:12 of EPTP).
- VM exits are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

30.5.6 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 11.10.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. VM exits clear any address-range monitoring that may be in effect.

30.6 LOADING MSRS

VM exits may load MSRs from the VM-exit MSR-load area (see Section 27.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101H (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 0000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be loaded on VM exits for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.
- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.¹

If processing fails for any entry, a VMX abort occurs. See Section 30.7.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM exit, the logical processor does not use any translations that were cached before the transition.

1. Note the following about processors that support Intel 64 architecture. If CR0.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. Since CR0.PG is always 1 in VMX operation, the IA32_EFER MSR should not be included in the VM-exit MSR-load area for the purpose of modifying the LME bit.

30.7 VMX ABORTS

A problem encountered during a VM exit leads to a **VMX abort**. A VMX abort takes a logical processor into a shut-down state as described below.

A VMX abort does not modify the VMCS data in the VMCS region of any active VMCS. The contents of these data are thus suspect after the VMX abort.

On a VMX abort, a logical processor saves a nonzero 32-bit VMX-abort indicator field at byte offset 4 in the VMCS region of the VMCS whose misconfiguration caused the failure (see Section 27.2). The following values are used:

1. There was a failure in saving guest MSRs (see Section 30.4).
2. Host checking of the page-directory-pointer-table entries (PDPTes) failed (see Section 30.5.4).
3. The current VMCS has been corrupted (through writes to the corresponding VMCS region) in such a way that the logical processor cannot complete the VM exit properly.
4. There was a failure on loading host MSRs (see Section 30.6).
5. There was a machine-check event during VM exit (see Section 30.8).
6. The logical processor was in IA-32e mode before the VM exit and the “host address-space size” VM-exit control was 0 (see Section 30.5).

Some of these causes correspond to failures during the loading of state from the host-state area. Because the loading of such state may be done in any order (see Section 30.5) a VM exit that might lead to a VMX abort for multiple reasons (for example, the current VMCS may be corrupt and the host PDPTes might not be properly configured). In such cases, the VMX-abort indicator could correspond to any one of those reasons.

A logical processor never reads the VMX-abort indicator in a VMCS region and writes it only with one of the non-zero values mentioned above. The VMX-abort indicator allows software on one logical processor to diagnose the VMX-abort on another. For this reason, it is recommended that software running in VMX root operation zero the VMX-abort indicator in the VMCS region of any VMCS that it uses.

After saving the VMX-abort indicator, operation of a logical processor experiencing a VMX abort depends on whether the logical processor is in SMX operation:¹

- If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000DH, indicating “VMX abort.” See the Intel® Trusted Execution Technology Measured Launched Environment Programming Guide.
- If the logical processor is outside SMX operation, it issues a special bus cycle (to notify the chipset) and enters the **VMX-abort shutdown state**. RESET is the only event that wakes a logical processor from the VMX-abort shutdown state. The following events do not affect a logical processor in this state: machine-check events; INIT signals; external interrupts; non-maskable interrupts (NMIs); start-up IPIs (SIPIs); and system-management interrupts (SMIs).

30.8 MACHINE-CHECK EVENTS DURING VM EXIT

If a machine-check event occurs during VM exit, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM exit:
 - If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:¹
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine-check condition.”
 - If the logical processor is outside SMX operation, it goes to the shutdown state.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 7, “Safer Mode Extensions Reference,” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

- If CR4.MCE = 1, a machine-check exception (#MC) is generated:
 - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered to the guest normally.
 - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- The machine-check event is handled after VM exit completes:
 - If the VM exit ends with CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If the VM exit ends with CR4.MCE = 1, a machine-check exception (#MC) is delivered to the host normally.
- A VMX abort is generated (see Section 30.7). The logical processor blocks events as done normally in VMX abort. The VMX abort indicator is 5, for “machine-check event during VM exit.”

The first option is not used if the machine-check event occurs after any host state has been loaded. The second option is used only if VM entry is able to load all host state.

30.9 USER-INTERRUPT RECOGNITION AFTER VM EXIT

A VM exit results in recognition of a pending user interrupt if it completes with CR4.UINTR = IA32_EFER.LMA = 1 and with UIRR ≠ 0; otherwise, no pending user interrupt is recognized.

25. Updates to Chapter 32, Volume 3C

Change bars and text show changes to Chapter 32 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- Updated pseudocode in Section 32.2.3, "Virtualizing User-Interrupt Notifications".
- Updated Section 32.4, "Virtualizing Memory-Mapped APIC Accesses" for FRED.
- Clarified footnote 2 on page 32-11 that delivery of an event can occur through IDT or with FRED.
- In Section 32.6, "Posted-Interrupt Processing", updated name of VM-exit interruption-information field to exiting event identification field.

CHAPTER 32

APIC VIRTUALIZATION AND VIRTUAL INTERRUPTS

The VMCS includes controls that enable the virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC).

When these controls are used, the processor will emulate many accesses to the APIC, track the state of the virtual APIC, and deliver virtual interrupts — all in VMX non-root operation without a VM exit.¹

The processor tracks the state of the virtual APIC using a virtual-APIC page identified by the virtual-machine monitor (VMM). Section 32.1 discusses the virtual-APIC page and how the processor uses it to track the state of the virtual APIC.

The following are the VM-execution controls relevant to APIC virtualization and virtual interrupts (see Section 27.6 for information about the locations of these controls):

- **Virtual-interrupt delivery.** This control enables the evaluation and delivery of pending virtual interrupts (Section 32.2). It also enables the emulation of writes (memory-mapped or MSR-based, as enabled) to the APIC registers that control interrupt prioritization.
- **Use TPR shadow.** This control enables emulation of accesses to the APIC's task-priority register (TPR) via CR8 (Section 32.3) and, if enabled, via the memory-mapped or MSR-based interfaces.
- **Virtualize APIC accesses.** This control enables virtualization of memory-mapped accesses to the APIC (Section 32.4) by causing VM exits on accesses to a VMM-specified APIC-access page. Some of the other controls, if set, may cause some of these accesses to be emulated rather than causing VM exits.
- **Virtualize x2APIC mode.** This control enables virtualization of MSR-based accesses to the APIC (Section 32.5).
- **APIC-register virtualization.** This control allows memory-mapped and MSR-based reads of most APIC registers (as enabled) by satisfying them from the virtual-APIC page. It directs memory-mapped writes to the APIC-access page to the virtual-APIC page, following them by VM exits for VMM emulation.
- **Process posted interrupts.** This control allows software to post virtual interrupts in a data structure and send a notification to another logical processor; upon receipt of the notification, the target processor will process the posted interrupts by copying them into the virtual-APIC page (Section 32.6).
- **IPI virtualization.** This control enables the virtualization of interprocessor interrupts (Section 32.1.6).

"Virtualize APIC accesses", "virtualize x2APIC mode", "virtual-interrupt delivery", and "APIC-register virtualization" are all secondary processor-based VM-execution controls; if bit 31 of the primary processor-based VM-execution controls is 0, the processor operates as if these controls were all 0. "IPI virtualization" is a tertiary processor-based VM-execution control; if bit 17 of the primary processor-based VM-execution controls is 0, the processor operates as if "IPI virtualization" were 0. See Section 27.6.2.

32.1 VIRTUAL APIC STATE

The **virtual-APIC page** is a 4-KByte region of memory that the processor uses to virtualize certain accesses to APIC registers and to manage virtual interrupts. The physical address of the virtual-APIC page is the **virtual-APIC address**, a 64-bit VM-execution control field in the VMCS (see Section 27.6.8).

Depending on the settings of certain VM-execution controls, the processor may virtualize certain fields on the virtual-APIC page with functionality analogous to that performed by the local APIC. Section 32.1.1 identifies and defines these fields. Section 32.1.2, Section 32.1.3, Section 32.1.4, and Section 32.1.5 detail the actions taken to virtualize updates to some of these fields.

With the exception of fields corresponding to virtualized APIC registers (defined in Section 32.1.1), software may modify the virtual-APIC page referenced by the current VMCS of a logical processor in VMX non-root operation. (This is an exception to the general requirement given in Section 27.11.4.)

1. In most cases, it is not necessary for a virtual-machine monitor (VMM) to inject virtual interrupts as part of VM entry.

32.1.1 Virtualized APIC Registers

Depending on the setting of certain VM-execution controls, a logical processor may virtualize certain accesses to APIC registers using the following fields on the virtual-APIC page:

- **Virtual task-priority register (VTPR):** the 32-bit field located at offset 080H on the virtual-APIC page.
- **Virtual processor-priority register (VPPR):** the 32-bit field located at offset 0A0H on the virtual-APIC page.
- **Virtual end-of-interrupt register (VEOI):** the 32-bit field located at offset 0B0H on the virtual-APIC page.
- **Virtual interrupt-service register (VISR):** the 256-bit value comprising eight non-contiguous 32-bit fields at offsets 100H, 110H, 120H, 130H, 140H, 150H, 160H, and 170H on the virtual-APIC page. Bit x of the VISR is at bit position $(x \& 1FH)$ at offset $(100H \mid ((x \& E0H) \gg 1))$. The processor uses only the low 4 bytes of each of the 16-byte fields at offsets 100H, 110H, 120H, 130H, 140H, 150H, 160H, and 170H.
- **Virtual interrupt-request register (VIRR):** the 256-bit value comprising eight non-contiguous 32-bit fields at offsets 200H, 210H, 220H, 230H, 240H, 250H, 260H, and 270H on the virtual-APIC page. Bit x of the VIRR is at bit position $(x \& 1FH)$ at offset $(200H \mid ((x \& E0H) \gg 1))$. The processor uses only the low 4 bytes of each of the 16-Byte fields at offsets 200H, 210H, 220H, 230H, 240H, 250H, 260H, and 270H.
- **Virtual interrupt-command register (VICR_LO):** the 32-bit field located at offset 300H on the virtual-APIC page.
- **Virtual interrupt-command register (VICR_HI):** the 32-bit field located at offset 310H on the virtual-APIC page.

The VTPR field virtualizes the TPR whenever the “use TPR shadow” VM-execution control is 1. The other fields indicated above virtualize the corresponding APIC registers whenever the “virtual-interrupt delivery” VM-execution control is 1. (VICR_LO and VICR_HI also virtualize the ICR when the “IPI virtualization” VM-execution control is 1.)

32.1.2 TPR Virtualization

The processor performs **TPR virtualization** in response to the following operations: (1) virtualization of the MOV to CR8 instruction; (2) virtualization of a write to offset 080H on the APIC-access page; and (3) virtualization of the WRMSR instruction with ECX = 808H. See Section 32.3, Section 32.4.3, and Section 32.5 for details of when TPR virtualization is performed.

The following pseudocode details the behavior of TPR virtualization:

```

IF “virtual-interrupt delivery” is 0
    THEN
        IF VTPR[7:4] < TPR threshold (see Section 27.6.8)
            THEN cause VM exit due to TPR below threshold;
        FI;
    ELSE
        perform PPR virtualization (see Section 32.1.3);
        evaluate pending virtual interrupts (see Section 32.2.1);
    FI;

```

Any VM exit caused by TPR virtualization is trap-like: the instruction causing TPR virtualization completes before the VM exit occurs (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).

32.1.3 PPR Virtualization

The processor performs **PPR virtualization** in response to the following operations: (1) VM entry; (2) TPR virtualization; and (3) EOI virtualization. See Section 29.3.2.5, Section 32.1.2, and Section 32.1.4 for details of when PPR virtualization is performed.

PPR virtualization uses the guest interrupt status (specifically, SVI; see Section •) and VTPR. The following pseudocode details the behavior of PPR virtualization:

```

IF VTPR[7:4] ≥ SVI[7:4]

```

```

    THEN VPPR := VTPR & FFH;
    ELSE VPPR := SVI & FOH;
FI;

```

PPR virtualization always clears bytes 3:1 of VPPR.

PPR virtualization is caused only by TPR virtualization, EOI virtualization, and VM entry. Delivery of a virtual interrupt also modifies VPPR, but in a different way (see Section 32.2.2). No other operations modify VPPR, even if they modify SVI, VISR, or VTPR.

32.1.4 EOI Virtualization

The processor performs **EOI virtualization** in response to the following operations: (1) virtualization of a write to offset 0B0H on the APIC-access page; and (2) virtualization of the WRMSR instruction with ECX = 80BH. See Section 32.4.3 and Section 32.5 for details of when EOI virtualization is performed. EOI virtualization occurs only if the “virtual-interrupt delivery” VM-execution control is 1.

EOI virtualization uses and updates the guest interrupt status (specifically, SVI; see Section •). The following pseudocode details the behavior of EOI virtualization:

```

Vector := SVI;
VISR[Vector] := 0; (see Section 32.1.1 for definition of VISR)
IF any bits set in VISR
    THEN SVI := highest index of bit set in VISR
    ELSE SVI := 0;
FI;
perform PPR virtualization (see Section 32.1.3);
IF EOI_exit_bitmap[Vector] = 1 (see Section 27.6.8 for definition of EOI_exit_bitmap)
    THEN cause EOI-induced VM exit with Vector as exit qualification;
    ELSE evaluate pending virtual interrupts; (see Section 32.2.1)
FI;

```

Any VM exit caused by EOI virtualization is trap-like: the instruction causing EOI virtualization completes before the VM exit occurs (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).

32.1.5 Self-IPI Virtualization

The processor performs **self-IPI virtualization** in response to the following operations: (1) virtualization of a write to offset 300H on the APIC-access page; and (2) virtualization of the WRMSR instruction with ECX = 83FH. See Section 32.4.3 and Section 32.5 for details of when self-IPI virtualization is performed. Self-IPI virtualization occurs only if the “virtual-interrupt delivery” VM-execution control is 1.

Each operation that leads to self-IPI virtualization provides an 8-bit vector (see Section 32.4.3 and Section 32.5). Self-IPI virtualization updates the guest interrupt status (specifically, RVI; see Section •). The following pseudocode details the behavior of self-IPI virtualization:

```

VIRR[Vector] := 1; (see Section 32.1.1 for definition of VIRR)
RVI := max{RVI, Vector};
evaluate pending virtual interrupts; (see Section 32.2.1)

```

32.1.6 IPI Virtualization

The processor performs **IPI virtualization** in response to the following operations: (1) virtualization of a write to offset 300H on the APIC-access page (Section 32.4.3); (2) virtualization of the WRMSR instruction with ECX = 830H (Section 32.5); and (3) virtualization of some executions of SENDUIPI (Section 32.7). IPI virtualization occurs only if the “IPI virtualization” VM-execution control is 1.

IPI virtualization uses virtual-interrupt posting, which is described in Section 32.1.6.1. Section 32.1.6.2 gives the details of the operation of IPI virtualization.

32.1.6.1 Virtual-Interrupt Posting

IPI virtualization is based on **virtual-interrupt posting**, a process that can direct virtual interrupts to a specific virtual processor. With virtual-interrupt posting, a hardware or software agent “posts” a virtual interrupt in a data structure (**posted-interrupt descriptor** or **PID**) and then sends an interrupt (notification) to the logical processor on which the target virtual processor is operating. When that logical processor receives the notification, it uses information in the PID to deliver the virtual interrupt to the virtual processor (see Section 32.6).

A PID is a 64-byte data structure. In an expected usage, there is one PID for each virtual processor; the virtual processor’s VMCS contains a pointer to its PID. A PID has the format shown in Table 32-1.

Table 32-1. Format of Posted-Interrupt Descriptor (PID)

Bit Position(s)	Name	Description
255:0	Posted-interrupt requests (PIR)	One bit for each interrupt vector. There is a posted-interrupt request for a vector if the corresponding bit is 1.
256	Outstanding notification (ON)	If this bit is set, there is a notification outstanding for one or more posted interrupts in bits 255:0.
257	Suppress notify (SN)	Setting this bit directs agents not to send notifications.
271:258	Reserved	Reserved.
279:272	Notify vector (NV)	Notifications will use this vector.
287:280	Reserved	Reserved.
319:288	Notify destination (NDST)	Notifications will be directed to this physical APIC ID.
511:320	Reserved	Reserved.

A hardware or software agent posts a virtual interrupt to a virtual processor with following steps:

1. Read the PIR field in the virtual processor’s PID and write it back atomically, setting the bit that corresponds to the virtual interrupt’s vector.
2. Read the notification-information field in the PID and write it back atomically, setting the ON bit if the ON and SN bits were both 0 in the value read. (Step #2 may be done atomically with step #1.)
3. If step #2 changed the ON bit from 0 to 1, send a notification. The notification is an ordinary interrupt sent to the physical APIC ID NDST with vector NV.

A processor’s response to the delivery of a notification is detailed in Section 32.6.

The use of virtual-interrupt posting for IPI virtualization is explained in Section 32.1.6.2.

32.1.6.2 IPI Virtualization Using Virtual-Interrupt Posting

Each operation that leads to IPI virtualization provides an 8-bit virtual vector V and a 32-bit virtual APIC ID T. IPI virtualization uses those values to initiate the indicated virtual IPI using the **PID-pointer table**.

The PID-pointer table is a data structure referenced by the PID-pointer table address, a field in the VMCS. Each entry in the PID-pointer table contains the following information:

- Bits 63:6 contain bits 63:6 of the 64-bit physical address of a PID (see Section 32.1.6.1).
- Bits 5:1 are reserved and must be 0.
- Bit 0 is a valid bit.

Each such address must be 64-byte aligned. The index of the last entry in the table is also a field in the VMCS.

When virtualizing an IPI, the CPU uses the virtual APIC ID T to select an entry from the PID-pointer table. It uses the address in that entry to locate a posted-interrupt descriptor (PID) and then posts a virtual interrupt with vector V in that PID. The following pseudocode details the behavior of IPI virtualization:

```

IF V < 16
    THEN APIC-write VM exit;           // illegal vector
ELSE IF T ≤ last PID-pointer index    // a field in the VMCS

```

```

THEN
    PID_ADDR := 8 bytes at (PID-pointer table address + (T << 3));
    IF PID_ADDR sets bits beyond the processor's physical-address width1 OR
        PID_ADDR[5:0] ≠ 000001b // PID pointer not valid or reserved bits set
    THEN APIC-write VM exit; // See Section 32.4.3.3
    ELSE
        PID_ADDR[0] := 0; // clear the valid bit before using as an address
        PIR := 32 bytes at PID_ADDR; // under lock
        PIR[V] := 1;
        store PIR at PID_ADDR; // release lock; corresponds to step #1 in Section 32.1.6.1
        NotifyInfo := 8 bytes at PID_ADDR + 32; // under lock
        IF NotifyInfo.ON = 0 AND NotifyInfo.SN = 0
            THEN
                NotifyInfo.ON := 1;
                SendNotify := 1;
            ELSE SendNotify := 0;
        FI;
        store NotifyInfo at PID_ADDR + 32; // release lock; corresponds to step #2 in Section 32.1.6.1
        IF SendNotify = 1
            THEN send an IPI specified by NotifyInfo.NDST and NotifyInfo.NV; // step #3 in Section 32.1.6.1
        FI;
    FI;
ELSE APIC-write VM exit; // virtual APIC ID beyond end of tables
FI;

```

The sending of the notification IPI is indicated by fields in the selected PID: NDST (PID[319:288]) and NV (PID[279:272]):

- If the local APIC is in xAPIC mode, this is the IPI that would be generated by writing NDST[15:8] (PID[303:296]) to ICR_HI[31:24] (offset 310H from IA32_APIC_BASE) and then writing NV to ICR_LO (offset 300H from IA32_APIC_BASE).
- If the local APIC is in x2APIC mode, this is the IPI that would be generated by executing WRMSR with ECX = 830H (ICR), EAX = NV, and EDX = NDST.

If the pseudocode specifies an APIC-write VM exit, this VM exit occurs as if there had been a write access to page offset 300H on the APIC-access page (see Section 32.4.3.3).

32.2 EVALUATION AND DELIVERY OF VIRTUAL INTERRUPTS

If the “virtual-interrupt delivery” VM-execution control is 1, certain actions in VMX non-root operation or during VM entry cause the processor to evaluate and deliver virtual interrupts.

Evaluation of virtual interrupts is triggered by certain actions change the state of the virtual-APIC page and is described in Section 32.2.1. This evaluation may result in recognition of a virtual interrupt. Once a virtual interrupt is recognized, the processor may deliver it within VMX non-root operation without a VM exit. Virtual-interrupt delivery is described in Section 32.2.2.

32.2.1 Evaluation of Pending Virtual Interrupts

If the “virtual-interrupt delivery” VM-execution control is 1, certain actions cause a logical processor to **evaluate pending virtual interrupts**.

The following actions cause the evaluation of pending virtual interrupts: VM entry; TPR virtualization; EOI virtualization; self-IPI virtualization; and posted-interrupt processing. See Section 29.3.2.5, Section 32.1.2, Section 32.1.4, Section 32.1.5, and Section 32.6 for details of when evaluation of pending virtual interrupts is performed. No other operations cause the evaluation of pending virtual interrupts, even if they modify RVI or VPPR.

1. Usually, the processor's physical-address width is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), the width is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

Evaluation of pending virtual interrupts uses the guest interrupt status (specifically, RVI; see Section •). The following pseudocode details the evaluation of pending virtual interrupts:

```

IF logical processor in SEAM and RVI < 31
    THEN RVI := 0;
FI;
IF "interrupt-window exiting" is 0 AND
RVI[7:4] > VPPR[7:4] (see Section 32.1.1 for definition of VPPR)
    THEN recognize a pending virtual interrupt;
ELSE
    do not recognize a pending virtual interrupt;
FI;

```

Once recognized, a virtual interrupt may be delivered in VMX non-root operation; see Section 32.2.2. Note that, in SEAM, no virtual interrupt will be delivered with a vector lower than 31.

Evaluation of pending virtual interrupts is caused only by VM entry, TPR virtualization, EOI virtualization, self-IPI virtualization, and posted-interrupt processing. No other operations do so, even if they modify RVI or VPPR. The logical processor ceases recognition of a pending virtual interrupt following the delivery of a virtual interrupt.

32.2.2 Virtual-Interrupt Delivery

If a virtual interrupt has been recognized (see Section 32.2.1), it is delivered at an instruction boundary when the following conditions all hold: (1) RFLAGS.IF = 1; (2) there is no blocking by STI; (3) there is no blocking by MOV SS or by POP SS; and (4) the "interrupt-window exiting" VM-execution control is 0.

Virtual-interrupt delivery has the same priority as that of VM exits due to the 1-setting of the "interrupt-window exiting" VM-execution control.¹ Thus, non-maskable interrupts (NMIs) and higher priority events take priority over delivery of a virtual interrupt; delivery of a virtual interrupt takes priority over external interrupts and lower priority events.

Virtual-interrupt delivery wakes a logical processor from the same inactive activity states as would an external interrupt. Specifically, it wakes a logical processor from the states entered using the HLT and MWAIT instructions. It does not wake a logical processor in the shutdown state or in the wait-for-SIPI state.

Virtual-interrupt delivery updates the guest interrupt status (both RVI and SVI; see Section •) and delivers an event within VMX non-root operation without a VM exit. The following pseudocode details the behavior of virtual-interrupt delivery (see Section 32.1.1 for definition of VISR, VIRR, and VPPR):

```

Vector := RVI;
VISR[Vector] := 1;
SVI := Vector;
VPPR := Vector & FOH;
VIRR[Vector] := 0;
IF any bits set in VIRR
    THEN RVI := highest index of bit set in VIRR
    ELSE RVI := 0;
FI;
IF logical processor is in SEAM and RVI < 31
    THEN RVI := 0;
FI;
cease recognition of any pending virtual interrupt;
IF transactional execution is in effect
    THEN abort transactional execution and transition to a non-transactional execution;
FI;
IF logical processor is in enclave mode

```

1. A logical processor never recognizes or delivers a virtual interrupt if the "interrupt-window exiting" VM-execution control is 1. Because of this, the relative priority of virtual-interrupt delivery and VM exits due to the 1-setting of that control is not defined.

```

    THEN cause an Asynchronous Enclave Exit (AEX) (see Chapter 40, "Enclave Exiting Events")
FI;
IF CR4.UINTR = 1 AND IA32_EFER.LMA = 1 AND Vector = UINV
    THEN virtualize user-interrupt notification identification and processing (see Section 32.2.3)
    ELSE deliver interrupt with Vector to guest software;
FI;

```

32.2.3 Virtualizing User-Interrupt Notifications

Section 9.5 describes the process of user-interrupt notification identification and processing. If the "virtual-interrupt delivery" VM-execution control is 1, this process is modified as described in the following paragraphs.

The virtualized form of user-interrupt notification identification begins as described in Section 32.2.2. Following this, instead of writing zero to the EOI register in the local APIC, the logical processor performs the initial steps of EOI virtualization:

```

VISR[V] := 0;
IF any bit is set in VISR
    THEN SVI := highest index of bit set in VISR
    ELSE SVI := 0;
FI;
perform PPR virtualization (Section 32.1.3);

```

Unlike EOI virtualization resulting from a guest write to the EOI register (as defined for virtual-interrupt delivery), the logical processor does not check the EOI-exit bitmap as part of this modified form of user-interrupt notification identification, and the corresponding VM exits cannot occur.

Following this modified form of user-interrupt notification identification, the logical processor then performs user-interrupt notification processing as specified in Section 9.5.2.

A logical processor is not interruptible during this modified form of user-interrupt notification identification or between it and any subsequent user-interrupt notification processing.

If the user-interrupt notification identification that precedes user-interrupt notification processing occurred while the logical processor was in the HLT state, the logical processor returns to the HLT state following user-interrupt notification processing.

32.3 VIRTUALIZING CR8-BASED TPR ACCESSES

In 64-bit mode, software can access the local APIC's task-priority register (TPR) through CR8. Specifically, software uses the MOV from CR8 and MOV to CR8 instructions (see Section 13.8.6, "Task Priority in IA-32e Mode"). This section describes how these accesses can be virtualized.

A virtual-machine monitor can virtualize these CR8-based APIC accesses by setting the "CR8-load exiting" and "CR8-store exiting" VM-execution controls, ensuring that the accesses cause VM exits (see Section 28.1.3). Alternatively, there are methods for virtualizing some CR8-based APIC accesses without VM exits.

Normally, an execution of MOV from CR8 or MOV to CR8 that does not fault or cause a VM exit accesses the APIC's TPR. However, such an execution are treated specially if the "use TPR shadow" VM-execution control is 1. The following items provide details:

- **MOV from CR8.** The instruction loads bits 3:0 of its destination operand with bits 7:4 of VTPR (see Section 32.1.1). Bits 63:4 of the destination operand are cleared.
- **MOV to CR8.** The instruction stores bits 3:0 of its source operand into bits 7:4 of VTPR; the remainder of VTPR (bits 3:0 and bits 31:8) are cleared. Following this, the processor performs TPR virtualization (see Section 32.1.2).

32.4 VIRTUALIZING MEMORY-MAPPED APIC ACCESSES

When the local APIC is in xAPIC mode, software accesses the local APIC's control registers using a memory-mapped interface. Specifically, software uses linear addresses that translate to physical addresses on page frame indicated by the base address in the IA32_APIC_BASE MSR (see Section 13.4.4, "Local APIC Status and Location"). This section describes how these accesses can be virtualized.

A virtual-machine monitor (VMM) can virtualize these memory-mapped APIC accesses by ensuring that any access to a linear address that would access the local APIC instead causes a VM exit. This could be done using paging or the extended page-table mechanism (EPT). Another way is by using the 1-setting of the "virtualize APIC accesses" VM-execution control.

If the "virtualize APIC accesses" VM-execution control is 1, the logical processor treats specially memory accesses using linear addresses that translate to physical addresses in the 4-KByte **APIC-access page**.^{1,2} (The APIC-access page is identified by the **APIC-access address**, a field in the VMCS; see Section 27.6.8.)

In general, an access to the APIC-access page causes an **APIC-access VM exit**. APIC-access VM exits provide a VMM with information about the access causing the VM exit. Section 32.4.1 discusses the priority of APIC-access VM exits.

Certain VM-execution controls enable the processor to virtualize certain accesses to the APIC-access page without a VM exit. In general, this virtualization causes these accesses to be made to the virtual-APIC page instead of the APIC-access page.

NOTES

Unless stated otherwise, this section characterizes only linear accesses to the APIC-access page; an access to the APIC-access page is a linear access if (1) it results from a memory access using a linear address; and (2) the access's physical address is the translation of that linear address. Section 32.4.6 discusses accesses to the APIC-access page that are not linear accesses.

The distinction between the APIC-access page and the virtual-APIC page allows a VMM to share paging structures or EPT paging structures among the virtual processors of a virtual machine (the shared paging structures referencing the same APIC-access address, which appears in the VMCS of all the virtual processors) while giving each virtual processor its own virtual APIC (the VMCS of each virtual processor will have a unique virtual-APIC address).

Section 32.4.2 discusses when and how the processor may virtualize read accesses from the APIC-access page. Section 32.4.3 does the same for write accesses. When virtualizing a write to the APIC-access page, the processor typically takes actions in addition to passing the write through to the virtual-APIC page.

The discussion in those sections uses the concept of an **operation** within which these memory accesses may occur. For those discussions, an "operation" can be an iteration of a REP-prefixed string instruction, an execution of any other instruction, or delivery of an event through the IDT or with FRED.

The 1-setting of the "virtualize APIC accesses" VM-execution control may also affect accesses to the APIC-access page that do not result directly from linear addresses. This is discussed in Section 32.4.6.

Special treatment may apply to Intel SGX instructions or if the logical processor is in enclave mode. See Section 42.5.3 for details.

1. Even when addresses are translated using EPT (see Section 31.3), the determination of whether an APIC-access VM exit occurs depends on an access's physical address, not its guest-physical address. Even when CR0.PG = 0, ordinary memory accesses by software use linear addresses; the fact that CR0.PG = 0 means only that the identity translation is used to convert linear addresses to physical (or guest-physical) addresses.
2. If EPT is enabled and there is write to a guest-physical address that translates to an address on the APIC-access page that is eligible for sub-page write permissions (see Section 31.3.4.1), the processor may treat the write as if the "virtualize APIC accesses" VM-execution control were 0 (and not apply the treatment specified in this section). For that reason, it is recommended that software not configure any guest-physical address that translates to an address on the APIC-access page to be eligible for sub-page write permissions.

32.4.1 Priority of APIC-Access VM Exits

The following items specify the priority of APIC-access VM exits relative to other events.

- The priority of an APIC-access VM exit due to a memory access is below that of any page fault or EPT violation that that access may incur. That is, an access does not cause an APIC-access VM exit if it would cause a page fault or an EPT violation.
- A memory access does not cause an APIC-access VM exit until after the accessed flags are set in the paging structures (including EPT paging structures, if enabled).
- A write access does not cause an APIC-access VM exit until after the dirty flags are set in the appropriate paging structure and EPT paging structure (if enabled).
- With respect to all other events, any APIC-access VM exit due to a memory access has the same priority as any page fault or EPT violation that the access could cause. (This item applies to other events that the access may generate as well as events that may be generated by other accesses by the same operation.)

These principles imply, among other things, that an APIC-access VM exit may occur during the execution of a repeated string instruction (including `INS` and `OUTS`). Suppose, for example, that the first n iterations (n may be 0) of such an instruction do not access the APIC-access page and that the next iteration does access that page. As a result, the first n iterations may complete and be followed by an APIC-access VM exit. The instruction pointer saved in the VMCS references the repeated string instruction and the values of the general-purpose registers reflect the completion of n iterations.

32.4.2 Virtualizing Reads from the APIC-Access Page

A read access from the APIC-access page causes an APIC-access VM exit if any of the following are true:

- The “use TPR shadow” VM-execution control is 0.
- The access is for an instruction fetch.
- The access is more than 32 bits in size.
- The access is part of an operation for which the processor has already virtualized a write to the APIC-access page.
- The access is not entirely contained within the low 4 bytes of a naturally aligned 16-byte region. That is, bits 3:2 of the access’s address are 0, and the same is true of the address of the highest byte accessed.

If none of the above are true, whether a read access is virtualized depends on the setting of the “APIC-register virtualization” and “virtual-interrupt delivery” VM-execution controls:

- A read access is virtualized if its page offset is 080H (task priority) regardless of the settings of the “APIC-register virtualization” and “virtual-interrupt delivery” VM-execution controls.
- If the “virtual-interrupt delivery” VM-execution control is 1, a read access is virtualized if its page offset is 0B0H (end of interrupt) or 300H (interrupt command — low).
- If “APIC-register virtualization” is 1, a read access is virtualized if it is entirely within one of the following ranges of offsets:
 - 020H–023H (local APIC ID);
 - 030H–033H (local APIC version);
 - 080H–083H (task priority);
 - 0B0H–0B3H (end of interrupt);
 - 0D0H–0D3H (logical destination);
 - 0E0H–0E3H (destination format);
 - 0F0H–0F3H (spurious-interrupt vector);
 - 100H–103H, 110H–113H, 120H–123H, 130H–133H, 140H–143H, 150H–153H, 160H–163H, or 170H–173H (in-service);
 - 180H–183H, 190H–193H, 1A0H–1A3H, 1B0H–1B3H, 1C0H–1C3H, 1D0H–1D3H, 1E0H–1E3H, or 1F0H–1F3H (trigger mode);

- 200H–203H, 210H–213H, 220H–223H, 230H–233H, 240H–243H, 250H–253H, 260H–263H, or 270H–273H (interrupt request);
- 280H–283H (error status);
- 300H–303H or 310H–313H (interrupt command);
- 320H–323H, 330H–333H, 340H–343H, 350H–353H, 360H–363H, or 370H–373H (LVT entries);
- 380H–383H (initial count); or
- 3E0H–3E3H (divide configuration).

In all other cases, the access causes an APIC-access VM exit.

A read access from the APIC-access page that is virtualized returns data from the corresponding page offset on the virtual-APIC page.¹

32.4.3 Virtualizing Writes to the APIC-Access Page

Whether a write access to the APIC-access page is virtualized depends on the settings of the VM-execution controls and the page offset of the access. Section 32.4.3.1 details when APIC-write virtualization occurs.

Unlike reads, writes to the local APIC have side effects; because of this, virtualization of writes to the APIC-access page may require emulation specific to the access's page offset (which identifies the APIC register being accessed). Section 32.4.3.2 describes this **APIC-write emulation**.

For some page offsets, it is necessary for software to complete the virtualization after a write completes. In these cases, the processor causes an **APIC-write VM exit** to invoke VMM software. Section 32.4.3.3 discusses APIC-write VM exits.

32.4.3.1 Determining Whether a Write Access is Virtualized

A write access to the APIC-access page causes an APIC-access VM exit if any of the following are true:

- The "use TPR shadow" VM-execution control is 0.
- The access is more than 32 bits in size.
- The access is part of an operation for which the processor has already virtualized a write (with a different page offset or a different size) to the APIC-access page.
- The access is not entirely contained within the low 4 bytes of a naturally aligned 16-byte region. That is, bits 3:2 of the access's address are 0, and the same is true of the address of the highest byte accessed.

If none of the above are true, whether a write access is virtualized depends on the settings of the "APIC-register virtualization", "virtual-interrupt delivery", and "IPI virtualization" VM-execution controls:

- A write access is virtualized if its page offset is 080H (task priority) regardless of the settings of the "APIC-register virtualization" and "virtual-interrupt delivery" VM-execution controls.
- If the "virtual-interrupt delivery" VM-execution control is 1, a write access is virtualized if its page offset is 0B0H (end of interrupt) or 300H (interrupt command — low).
- If the "IPI virtualization" VM-execution control is 1, a write access is virtualized if its page offset is 300H.
- If the "APIC-register virtualization" VM-execution control is 1, a write access is virtualized if it is entirely within one the following ranges of offsets:
 - 020H–023H (local APIC ID);
 - 080H–083H (task priority);
 - 0B0H–0B3H (end of interrupt);
 - 0D0H–0D3H (logical destination);
 - 0E0H–0E3H (destination format);

1. The memory type used for accesses that read from the virtual-APIC page is reported in bits 53:50 of the IA32_VMX_BASIC MSR (see Appendix A.1).

- 0F0H–0F3H (spurious-interrupt vector);
- 280H–283H (error status);
- 300H–303H or 310H–313H (interrupt command);
- 320H–323H, 330H–333H, 340H–343H, 350H–353H, 360H–363H, or 370H–373H (LVT entries);
- 380H–383H (initial count); or
- 3E0H–3E3H (divide configuration).

In all other cases, the access causes an APIC-access VM exit.

The processor virtualizes a write access to the APIC-access page by writing data to the corresponding page offset on the virtual-APIC page.¹ Following this, the processor performs certain actions after completion of the operation of which the access was a part.² APIC-write emulation is described in Section 32.4.3.2.

32.4.3.2 APIC-Write Emulation

If the processor virtualizes a write access to the APIC-access page, it performs additional actions after completion of an operation of which the access was a part. These actions are called **APIC-write emulation**.

The details of APIC-write emulation depend upon the page offset of the virtualized write access:³

- 080H (task priority). The processor clears bytes 3:1 of VTPR and then causes TPR virtualization (Section 32.1.2).
- 0B0H (end of interrupt). If the “virtual-interrupt delivery” VM-execution control is 1, the processor clears VEOI and then causes EOI virtualization (Section 32.1.4); otherwise, the processor causes an APIC-write VM exit (Section 32.4.3.3).
- 300H (interrupt command — low). If the “virtual-interrupt delivery” VM-execution control is 1, the processor checks the value of VICR_LO to determine whether the following are all true:
 - Reserved bits (31:20, 17:16, 13) and bit 12 (delivery status) are all 0.
 - Bits 19:18 (destination shorthand) are 01B (self).
 - Bit 15 (trigger mode) is 0 (edge).
 - Bits 10:8 (delivery mode) are 000B (fixed).
 - Bits 7:4 (the upper half of the vector) are **not** 0000B.

If all of the items above are true, the processor performs self-IPI virtualization using the 8-bit vector in byte 0 of VICR_LO (Section 32.1.5).

If the “virtual-interrupt delivery” VM-execution control is 0, or if any of the items above are false, behavior depends on the setting of the “IPI virtualization” VM-execution control:

- If the “IPI virtualization” VM-execution control is 1, the processor checks the value of VICR_LO to determine whether the following are all true:
 - Reserved bits (31:20, 17:16, 13) and bit 12 (delivery status) are all 0.
 - Bits 19:18 (destination shorthand) are 00B (no shorthand).
 - Bit 15 (trigger mode) is 0 (edge).
 - Bit 11 (destination mode) is 0 (physical).
 - Bits 10:8 (delivery mode) are 000B (fixed).

1. The memory type used for accesses that write to the virtual-APIC page is reported in bits 53:50 of the IA32_VMX_BASIC MSR (see Appendix A.1).

2. Recall that, for the purposes of this discussion, an operation is an iteration of a REP-prefixed string instruction, an execution of any other instruction, or delivery of an event through the IDT [or with FRED](#).

3. For any operation, there can be only one page offset for which a write access was virtualized. This is because a write access is not virtualized if the processor has already virtualized a write access for the same operation with a different page offset.

If all of the items above are true, the processor performs IPI virtualization using the 8-bit vector in byte 0 of VICR_LO and the 8-bit APIC ID in VICR_HI[31:24] (Section 32.1.6); otherwise, the processor causes an APIC-write VM exit.

- If the “IPI virtualization” VM-execution control is 0, the processor causes an APIC-write VM exit.
- 310H–313H (interrupt command — high). The processor clears bytes 2:0 of VICR_HI. No other virtualization or VM exit occurs.
- Any other page offset. The processor causes an APIC-write VM exit.

APIC-write emulation takes priority over system-management interrupts (SMIs), INIT signals, and lower priority events. APIC-write emulation is not blocked if RFLAGS.IF = 0 or by the MOV SS, POP SS, or STI instructions.

If an operation causes a fault after a write access to the APIC-access page and before APIC-write emulation, and that fault is delivered without a VM exit, APIC-write emulation occurs after the fault is delivered and before the fault handler can execute. If an operation causes a VM exit (perhaps due to a fault) after a write access to the APIC-access page and before APIC-write emulation, the APIC-write emulation does not occur.

32.4.3.3 APIC-Write VM Exits

In certain cases, VMM software must be invoked to complete the virtualization of a write access to the APIC-access page. In this case, APIC-write emulation causes an **APIC-write VM exit**. (Section 32.4.3.2 details the cases that causes APIC-write VM exits.)

APIC-write VM exits are invoked by APIC-write emulation, and APIC-write emulation occurs after an operation that performs a write access to the APIC-access page. Because of this, every APIC-write VM exit is trap-like: it occurs after completion of the operation containing the write access that caused the VM exit (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).

The basic exit reason for an APIC-write VM exit is “APIC write.” The exit qualification is the page offset of the write access that led to the VM exit.

As noted in Section 32.5, execution of WRMSR with ECX = 83FH (self-IPI MSR) can lead to an APIC-write VM exit if the “virtual-interrupt delivery” VM-execution control is 1; the exit qualification for the APIC-write VM exit is 3F0H. As noted in Section 32.1.6 and in Section 32.7, IPI virtualization and execution of SENDUIPI may lead to APIC-write VM exits; these VM exits produce an exit qualification of 300H.

32.4.4 Instruction-Specific Considerations

Certain instructions that use linear address may cause page faults even though they do not use those addresses to access memory. The APIC-virtualization features may affect these instructions as well:

- **CLFLUSH, CLFLUSHOPT.** With regard to faulting, the processor operates as if each of these instructions reads from the linear address in its source operand. If that address translates to one on the APIC-access page, the instruction may cause an APIC-access VM exit. If it does not, it will flush the corresponding cache line on the virtual-APIC page instead of the APIC-access page.
- **ENTER.** With regard to faulting, the processor operates if ENTER writes to the byte referenced by the final value of the stack pointer (even though it does not if its size operand is non-zero). If that value translates to an address on the APIC-access page, the instruction may cause an APIC-access VM exit. If it does not, it will cause the APIC-write emulation appropriate to the address’s page offset.
- **MASKMOVQ and MASKMOVDQU.** Even if the instruction’s mask is zero, the processor may operate with regard to faulting as if MASKMOVQ or MASKMOVDQU writes to memory (the behavior is implementation-specific). In such a situation, an APIC-access VM exit may occur.
- **MONITOR.** With regard to faulting, the processor operates as if MONITOR reads from the effective address in RAX. If the resulting linear address translates to one on the APIC-access page, the instruction may cause an

APIC-access VM exit.¹ If it does not, it will monitor the corresponding address on the virtual-APIC page instead of the APIC-access page.

- **PREFETCH.** An execution of the PREFETCH instruction that would result in an access to the APIC-access page does not cause an APIC-access VM exit. Such an access may prefetch data; if so, it is from the corresponding address on the virtual-APIC page.

Virtualization of accesses to the APIC-access page is principally intended for basic instructions such as AND, MOV, OR, TEST, XCHG, and XOR. Use of an instruction that normally operates on floating-point, SSE, AVX, or AVX-512 registers may cause an APIC-access VM exit unconditionally regardless of the page offset it accesses on the APIC-access page.

32.4.5 Issues Pertaining to Page Size and TLB Management

The 1-setting of the “virtualize APIC accesses” VM-execution is guaranteed to apply only if translations to the APIC-access address use a 4-KByte page. The following items provide details:

- If EPT is not in use, any linear address that translates to an address on the APIC-access page should use a 4-KByte page. Any access to a linear address that translates to the APIC-access page using a larger page may operate as if the “virtualize APIC accesses” VM-execution control were 0.
- If EPT is in use, any guest-physical address that translates to an address on the APIC-access page should use a 4-KByte page. Any access to a linear address that translates to a guest-physical address that in turn translates to the APIC-access page using a larger page may operate as if the “virtualize APIC accesses” VM-execution control were 0. (This is true also for guest-physical accesses to the APIC-access page; see Section 32.4.6.1.)

In addition, software should perform appropriate TLB invalidation when making changes that may affect APIC-virtualization. The specifics depend on whether VPIDs or EPT is being used:

- **VPIDs being used but EPT not being used.** Suppose that there is a VPID that has been used before and that software has since made either of the following changes: (1) set the “virtualize APIC accesses” VM-execution control when it had previously been 0; or (2) changed the paging structures so that some linear address translates to the APIC-access address when it previously did not. In that case, software should execute INVVPID (see “INVVPID— Invalidate Translations Based on VPID” in Section 33.3) before performing on the same logical processor and with the same VPID.²
- **EPT being used.** Suppose that there is an EPTP value that has been used before and that software has since made either of the following changes: (1) set the “virtualize APIC accesses” VM-execution control when it had previously been 0; or (2) changed the EPT paging structures so that some guest-physical address translates to the APIC-access address when it previously did not. In that case, software should execute INVEPT (see “INVEPT— Invalidate Translations Derived from EPT” in Section 33.3) before performing on the same logical processor and with the same EPTP value.³
- **Neither VPIDs nor EPT being used.** No invalidation is required.

Failure to perform the appropriate TLB invalidation may result in the logical processor operating as if the “virtualize APIC accesses” VM-execution control were 0 in responses to accesses to the affected address. (No invalidation is necessary if neither VPIDs nor EPT is being used.)

32.4.6 APIC Accesses Not Directly Resulting From Linear Addresses

Section 32.4 has described the treatment of accesses that use linear addresses that translate to addresses on the APIC-access page. This section considers memory accesses that do not result directly from linear addresses.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.
2. INVVPID should use either (1) the all-contexts INVVPID type; (2) the single-context INVVPID type with the VPID in the INVVPID descriptor; or (3) the individual-address INVVPID type with the linear address and the VPID in the INVVPID descriptor.
3. INVEPT should use either (1) the global INVEPT type; or (2) the single-context INVEPT type with the EPTP value in the INVEPT descriptor.

- An access is called a **guest-physical access** if (1) CR0.PG = 1;¹ (2) the “enable EPT” VM-execution control is 1;² (3) the access’s physical address is the result of an EPT translation; and (4) either (a) the access was not generated by a linear address; or (b) the access’s guest-physical address is not the translation of the access’s linear address. Section 32.4.6.1 discusses the treatment of guest-physical accesses to the APIC-access page.
- An access is called a **physical access** if (1) either (a) the “enable EPT” VM-execution control is 0; or (b) the access’s physical address is not the result of a translation through the EPT paging structures; and (2) either (a) the access is not generated by a linear address; or (b) the access’s physical address is not the translation of its linear address. Section 32.4.6.2 discusses the treatment of physical accesses to the APIC-access page.

32.4.6.1 Guest-Physical Accesses to the APIC-Access Page

Guest-physical accesses include the following when guest-physical addresses are being translated using EPT:

- Reads from the guest paging structures when translating a linear address (such an access uses a guest-physical address that is not the translation of that linear address).
- Loads of the page-directory-pointer-table entries by MOV to CR when the logical processor is using (or that causes the logical processor to use) PAE paging (see Section 5.4).
- Updates to the accessed and dirty flags in the guest paging structures when using a linear address (such an access uses a guest-physical address that is not the translation of that linear address).
- Memory accesses by Intel Processor Trace when the “Intel PT uses guest physical addresses” VM-execution control is 1 (see Section 28.5.4).

Every guest-physical access using a guest-physical address that translates to an address on the APIC-access page causes an APIC-access VM exit. Such accesses are never virtualized regardless of the page offset.

The following items specify the priority relative to other events of APIC-access VM exits caused by guest-physical accesses to the APIC-access page.

- The priority of an APIC-access VM exit caused by a guest-physical access to memory is below that of any EPT violation that that access may incur. That is, a guest-physical access does not cause an APIC-access VM exit if it would cause an EPT violation.
- With respect to all other events, any APIC-access VM exit caused by a guest-physical access has the same priority as any EPT violation that the guest-physical access could cause.

32.4.6.2 Physical Accesses to the APIC-Access Page

Physical accesses include the following:

- If the “enable EPT” VM-execution control is 0:
 - Reads from the paging structures when translating a linear address.
 - Loads of the page-directory-pointer-table entries by MOV to CR when the logical processor is using (or that causes the logical processor to use) PAE paging (see Section 5.4).
 - Updates to the accessed and dirty flags in the paging structures.
- If the “enable EPT” VM-execution control is 1, accesses to the EPT paging structures (including updates to the accessed and dirty flags for EPT).
- Any of the following accesses made by the processor to support VMX non-root operation:
 - Accesses to the VMCS region.
 - Accesses to data structures referenced (directly or indirectly) by physical addresses in VM-execution control fields in the VMCS. These include the I/O bitmaps, the MSR bitmaps, and the virtual-APIC page.
- Accesses that effect transitions into and out of SMM.³ These include the following:

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
2. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “enable EPT” VM-execution control were 0. See Section 27.6.2.

- Accesses to SMRAM during SMI delivery and during execution of RSM.
- Accesses during SMM VM exits (including accesses to MSEG) and during VM entries that return from SMM.

A physical access to the APIC-access page may or may not cause an APIC-access VM exit. If it does not cause an APIC-access VM exit, it may access the APIC-access page or the virtual-APIC page. Physical write accesses to the APIC-access page may or may not cause APIC-write emulation or APIC-write VM exits.

The priority of an APIC-access VM exit caused by physical access is not defined relative to other events that the access may cause.

It is recommended that software not set the APIC-access address to any of the addresses used by physical memory accesses (identified above). For example, it should not set the APIC-access address to the physical address of any of the active paging structures if the “enable EPT” VM-execution control is 0.

32.5 VIRTUALIZING MSR-BASED APIC ACCESSES

When the local APIC is in x2APIC mode, software accesses the local APIC’s control registers using the MSR interface. Specifically, software uses the RDMSR and WRMSR instructions, setting ECX (identifying the MSR being accessed) to values in the range 800H–8FFH (see Section 13.12, “Extended XAPIC (x2APIC)”). This section describes how these accesses can be virtualized.

A virtual-machine monitor can virtualize these MSR-based APIC accesses by configuring the MSR bitmaps (see Section 27.6.9) to ensure that the accesses cause VM exits (see Section 28.1.3). Alternatively, there are methods for virtualizing some MSR-based APIC accesses without VM exits.

Normally, an execution of RDMSR or WRMSR that does not fault or cause a VM exit accesses the MSR indicated in ECX. However, such an execution treats some values of ECX in the range 800H–8FFH specially if the “virtualize x2APIC mode” VM-execution control is 1. The following items provide details:

- **RDMSR.** The instruction’s behavior depends on the setting of the “APIC-register virtualization” VM-execution control.
 - If the “APIC-register virtualization” VM-execution control is 0, behavior depends upon the value of ECX.
 - If ECX contains 808H (indicating the TPR MSR), the instruction reads the 8 bytes from offset 080H on the virtual-APIC page (VTPR and the 4 bytes above it) into EDX:EAX. This occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not x2APIC mode).
 - If ECX contains any other value in the range 800H–8FFH, the instruction operates normally. If the local APIC is in x2APIC mode and ECX indicates a readable APIC register, EDX and EAX are loaded with the value of that register. If the local APIC is not in x2APIC mode or ECX does not indicate a readable APIC register, a general-protection fault occurs.
 - If “APIC-register virtualization” is 1 and ECX contains a value in the range 800H–8FFH, the instruction reads the 8 bytes from offset X on the virtual-APIC page into EDX:EAX, where $X = (ECX \& FFH) \ll 4$. This occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not in x2APIC mode).
- **WRMSR.** The instruction’s behavior depends on the value of ECX and the setting of the “virtual-interrupt delivery” and “IPI virtualization” VM-execution controls.

Special processing applies in the following cases: (1) ECX contains 808H (indicating the TPR MSR); (2) ECX contains 80BH (indicating the EOI MSR) and the “virtual-interrupt delivery” VM-execution control is 1; (3) ECX contains 83FH (indicating the self-IPI MSR) and the “virtual-interrupt delivery” VM-execution control is 1; and (4) ECX contains 830H (indicating the ICR MSR) and the “IPI virtualization” VM-execution control is 1.

If special processing applies, no general-protection exception is produced due to the fact that the local APIC is in xAPIC mode. However, WRMSR does perform the normal reserved-bit checking:

- If ECX contains 808H or 83FH, a general-protection fault occurs if either EDX or EAX[31:8] is non-zero.
- If ECX contains 80BH, a general-protection fault occurs if either EDX or EAX is non-zero.

3. Technically, these accesses do not occur in VMX non-root operation. They are included here for clarity.

- If ECX contains 830H, a general-protection fault occurs if any of bits 31:20, 17:16, or 13 of EAX is non-zero. If there is no fault, WRMSR stores EDX:EAX at offset X on the virtual-APIC page, where $X = (ECX \& FFH) \ll 4$. Following this, the processor performs an operation depending on the value of ECX:
- If ECX contains 808H, the processor performs TPR virtualization (see Section 32.1.2).
- If ECX contains 80BH, the processor performs EOI virtualization (see Section 32.1.4).
- If ECX contains 83FH, the processor then checks the value of EAX[7:4] and proceeds as follows:
 - If the value is non-zero, the logical processor performs self-IPI virtualization with the 8-bit vector in EAX[7:0] (see Section 32.1.5).
 - If the value is zero, the logical processor causes an APIC-write VM exit as if there had been a write access to page offset 3F0H on the APIC-access page (see Section 32.4.3.3).
- If ECX contains 830H, the processor then checks the value of VICR to determine whether the following are all true:
 - Bits 19:18 (destination shorthand) are 00B (no shorthand).
 - Bit 15 (trigger mode) is 0 (edge).
 - Bit 12 (unused) is 0.
 - Bit 11 (destination mode) is 0 (physical).
 - Bits 10:8 (delivery mode) are 000B (fixed).

If all of the items above are true, the processor performs IPI virtualization using the 8-bit vector in byte 0 of VICR and the 32-bit APIC ID in VICR[63:32] (see Section 32.1.6). Otherwise, the logical processor causes an APIC-write VM exit (see Section 32.4.3.3).

If special processing does not apply, the instruction operates normally. If the local APIC is in x2APIC mode and ECX indicates a writable APIC register, the value in EDX:EAX is written to that register. If the local APIC is not in x2APIC mode or ECX does not indicate a writable APIC register, a general-protection fault occurs.

32.6 POSTED-INTERRUPT PROCESSING

Posted-interrupt processing is a feature by which a processor processes the virtual interrupts by recording them as pending on the virtual-APIC page.

Posted-interrupt processing is enabled by setting the “process posted interrupts” VM-execution control. The processing is performed in response to the arrival of an interrupt with the **posted-interrupt notification vector**. In response to such an interrupt, the processor processes virtual interrupts recorded in a data structure called a **posted-interrupt descriptor (PID)**. The posted-interrupt notification vector and the address of the PID are fields in the VMCS; see Section 27.6.8.

If the “process posted interrupts” VM-execution control is 1, a logical processor uses a 64-byte posted-interrupt descriptor located at the posted-interrupt descriptor address. Table 32-2 gives the format of a PID:¹

Table 32-2. Format of Posted-Interrupt Descriptor (PID)

Bit Position(s)	Name	Description
255:0	Posted-interrupt requests (PIR)	One bit for each interrupt vector. There is a posted-interrupt request for a vector if the corresponding bit is 1.
256	Outstanding notification (ON)	If this bit is set, there is a notification outstanding for one or more posted interrupts in bits 255:0.
511:257	Reserved or used for virtual-interrupt posting	Some of these bits are used by virtual-interrupt posting (Section 32.1.6.1). Posted-interrupt processing does not use or modify these bits.

1. Table 32-1 gives the same format; it is repeated here for the reader, omitting fields that are not used by posted-interrupt processing.

Use of the PID differs from that of other data structures that are referenced by pointers in a VMCS. There is a general requirement that software ensure that each such data structure is modified only when no logical processor with a current VMCS that references it is in VMX non-root operation. That requirement does not apply to the posted-interrupt descriptor. There is a requirement, however, that such modifications be done using locked read-modify-write instructions or other atomic operations.

If the “external-interrupt exiting” VM-execution control is 1, any unmasked external interrupt causes a VM exit (see Section 28.2). If the “process posted interrupts” VM-execution control is also 1, this behavior is changed and the processor handles an external interrupt as follows:¹

1. The local APIC is acknowledged; this provides the processor core with an interrupt vector, called here the **physical vector**.
2. If the physical vector equals the posted-interrupt notification vector, the logical processor continues to the next step. Otherwise, a VM exit occurs as it would normally due to an external interrupt; the vector is saved in the **exiting-event identification** field.
3. The processor clears the outstanding-notification bit in the posted-interrupt descriptor. This is done atomically so as to leave the remainder of the descriptor unmodified (e.g., with a locked AND operation).
4. The processor writes zero to the EOI register in the local APIC; this dismisses the interrupt with the posted-interrupt notification vector from the local APIC.
5. The logical processor performs a logical-OR of PIR into VIRR and clears PIR. No other agent can read or write a PIR bit (or group of bits) between the time it is read (to determine what to OR into VIRR) and when it is cleared.
6. The logical processor sets RVI to be the maximum of the old value of RVI and the highest index of all bits that were set in PIR; if no bit was set in PIR, RVI is left unmodified.
7. The logical processor evaluates pending virtual interrupts as described in Section 32.2.1.

The logical processor performs the steps above in an uninterruptible manner. If step #7 leads to recognition of a virtual interrupt, the processor may deliver that interrupt immediately.

Steps #1 to #7 above occur when the interrupt controller delivers an unmasked external interrupt to the CPU core. The following items consider certain cases of interrupt delivery:

- Interrupt delivery can occur between iterations of a REP-prefixed instruction (after at least one iteration has completed but before all iterations have completed). If this occurs, the following items characterize processor state after posted-interrupt processing completes and before guest execution resumes:
 - RIP references the REP-prefixed instruction;
 - RCX, RSI, and RDI are updated to reflect the iterations completed; and
 - RFLAGS.RF = 1.
- Interrupt delivery can occur when the logical processor is in the active, HLT, or MWAIT states. If the logical processor had been in the active or MWAIT state before the arrival of the interrupt, it is in the active state following completion of step #7; if it had been in the HLT state, it returns to the HLT state after step #7 (if a pending virtual interrupt was recognized, the logical processor may immediately wake from the HLT state).
- Interrupt delivery can occur while the logical processor is in enclave mode. If the logical processor had been in enclave mode before the arrival of the interrupt, an Asynchronous Enclave Exit (AEX) may occur before the steps #1 to #7 (see Chapter 40, “Enclave Exiting Events”). If no AEX occurs before step #1 and a VM exit occurs at step #2, an AEX occurs before the VM exit is delivered.

1. VM entry ensures that the “process posted interrupts” VM-execution control is 1 only if the “external-interrupt exiting” VM-execution control is also 1. See Section 29.2.1.1.

32.7 VIRTUALIZING SENDUIPI

The user-interrupt feature includes the SENDUIPI instruction that software operating with CPL = 3 can use to send user interrupts to another software thread (“user IPIs”). The SENDUIPI instruction has the following high-level operation:

- read selected entry from user-interrupt target table;
- use address in entry to read the referenced user posted-interrupt descriptor (UPID);
- update certain fields in UPID;
- if necessary, send ordinary IPI indicated in UPID’s notification information;

The last step uses two fields in the UPID: an 8-bit notification vector (UPID.NV) and a 32-bit notification destination (an APIC ID, UPID.NDST). Outside of VMX non-root operation, the processor implements the last step as follows:

- If the local APIC is in xAPIC mode, it writes UPID.NDST[15:8] to ICR_HI[31:24] (offset 310H from IA32_APIC_BASE) and then writes UPID.NV to ICR_LO (offset 300H).
- If the local APIC is in x2APIC mode, it performs the control-register write that would be done by an execution of WRMSR with ECX = 310H (ICR), EAX = UPID.NV, and EDX = UPID.NDST.

In VMX non-root operation, implementation of the step depends on the settings of the “use TPR shadow,” “virtualize APIC accesses,” and “IPI virtualization” VM-execution controls:¹

1. If the “use TPR shadow” VM-execution control is 0, the behavior is not modified: the logical processor sends the specified IPI by writing to the local APIC’s ICR as specified above (based on the current mode of the local APIC).
2. If the “use TPR shadow” VM-execution control is 1 and the “virtualize APIC accesses” VM-execution control is 0, the logical processor virtualizes the sending of an x2APIC-mode IPI with the following steps:
 - a. The 64-bit value Z is written to offset 300H on the virtual-APIC page (VICR), where Z[7:0] = UPID.NV (the 8-bit virtual vector), Z[63:32] = UPID.NDST (the 32-bit virtual APIC ID) and Z[31:8] = 000000H (indicating a physically addressed fixed-mode IPI).
 - b. If the “IPI virtualization” VM-execution control is 1, IPI virtualization (Section 32.1.6) is performed using the vector UPID.NV and the 32-bit virtual APIC ID UPID.NDST.
3. If the “use TPR shadow” and “virtualize APIC accesses” VM-execution controls are both 1, the logical processor virtualizes the sending of an xAPIC-mode IPI by performing the following steps:
 - a. The 32-bit value X is written to offset 310H on the virtual-APIC page (VICR_HI), where X[31:24] = UPID.NDST[15:8] (the 8-bit virtual APIC ID) and X[23:0] = 000000H.²
 - b. The 32-bit value Y is written to offset 300H on the virtual-APIC page (VICR_LO), where Y[7:0] = UPID.NV (the 8-bit virtual vector) and Y[31:8] = 000000H (indicating a physically addressed fixed-mode IPI).
 - c. If the “IPI virtualization” VM-execution control is 1, IPI virtualization is performed using the vector UPID.NV and the APIC ID UPID.NDST[15:8]. IPI virtualization will use only the 8-bit APIC ID from bits 15:8 of the UPID’s destination field (the 8-bit value written earlier to bits 31:24 of VICR_HI).
4. If the “use TPR shadow” VM-execution control is 1 and the “IPI virtualization” VM-execution control is 0, an APIC-write VM exit occurs as if there had been a write access to page offset 300H on the APIC-access page (see Section 32.4.3.3).

1. The setting of the “virtualize x2APIC mode” VM-execution control does not affect this operation.

2. For xAPIC mode (which is virtualized if the “virtualize APIC accesses” VM-execution control is 1), the destination APIC ID is in byte 1 (**not** byte 0) of the UPID’s 4-byte NDST field.

26. Updates to Chapter 34, Volume 3C

Change bars and text show changes to Chapter 34 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- Added clarification for FRED event delivery in notes of Section 34.3.2, "Exiting From SMM".
- Added CR4.FRED to the list of situations when a shutdown occurs in Section 34.3.2.
- Updated Section 34.15.4.3, "Checks on VM-Entry Control Fields" for FRED.
- Documented FRED in CR4 in Section 34.15.6.5, "Loading Host State".

This chapter describes aspects of IA-64 and IA-32 architecture used in system management mode (SMM).

SMM provides an alternate operating environment that can be used to monitor and manage various system resources for more efficient energy usage, to control system hardware, and/or to run proprietary code. It was introduced into the IA-32 architecture in the Intel386 SL processor (a mobile specialized version of the Intel386 processor). It is also available in the Pentium M, Pentium 4, Intel Xeon, P6 family, and Pentium and Intel486 processors (beginning with the enhanced versions of the Intel486 SL and Intel486 processors).

34.1 SYSTEM MANAGEMENT MODE OVERVIEW

SMM is a special-purpose operating mode provided for handling system-wide functions like power management, system hardware control, or proprietary OEM-designed code. It is intended for use only by system firmware, not by applications software or general-purpose systems software. The main benefit of SMM is that it offers a distinct and easily isolated processor environment that operates transparently to the operating system or executive and software applications.

When SMM is invoked through a system management interrupt (SMI), the processor saves the current state of the processor (the processor's context), then switches to a separate operating environment defined by a new address space. The system management software executive (SMI handler) starts execution in that environment, and the critical code and data of the SMI handler reside in a physical memory region (SMRAM) within that address space. While in SMM, the processor executes SMI handler code to perform operations such as powering down unused disk drives or monitors, executing proprietary code, or placing the whole system in a suspended state. When the SMI handler has completed its operations, it executes a resume (RSM) instruction. This instruction causes the processor to reload the saved context of the processor, switch back to protected or real mode, and resume executing the interrupted application or operating-system program or task.

The following SMM mechanisms make it transparent to applications programs and operating systems:

- The only way to enter SMM is by means of an SMI.
- The processor executes SMM code in a separate address space that can be made inaccessible from the other operating modes.
- Upon entering SMM, the processor saves the context of the interrupted program or task.
- All interrupts normally handled by the operating system are disabled upon entry into SMM.
- The RSM instruction can be executed only in SMM.

Section 34.3 describes transitions into and out of SMM. The execution environment after entering SMM is in real-address mode with paging disabled ($CR0.PE = CR0.PG = 0$). In this initial execution environment, the SMI handler can address up to 4 GBytes of memory and can execute all I/O and system instructions. Section 34.5 describes in detail the initial SMM execution environment for an SMI handler and operation within that environment. The SMI handler may subsequently switch to other operating modes while remaining in SMM.

NOTES

Software developers should be aware that, even if a logical processor was using the physical-address extension (PAE) mechanism (introduced in the P6 family processors) or was in IA-32e mode before an SMI, this will not be the case after the SMI is delivered. This is because delivery of an SMI disables paging (see Table 34-4). (This does not apply if the dual-monitor treatment of SMIs and SMM is active; see Section 34.15.)

34.1.1 System Management Mode and VMX Operation

Traditionally, SMM services system management interrupts and then resumes program execution (back to the software stack consisting of executive and application software; see Section 34.2 through Section 34.13).

A virtual machine monitor (VMM) using VMX can act as a host to multiple virtual machines and each virtual machine can support its own software stack of executive and application software. On processors that support VMX, virtual-machine extensions may use system-management interrupts (SMIs) and system-management mode (SMM) in one of two ways:

- **Default treatment.** System firmware handles SMIs. The processor saves architectural states and critical states relevant to VMX operation upon entering SMM. When the firmware completes servicing SMIs, it uses RSM to resume VMX operation.
- **Dual-monitor treatment.** Two VM monitors collaborate to control the servicing of SMIs: one VMM operates outside of SMM to provide basic virtualization in support for guests; the other VMM operates inside SMM (while in VMX operation) to support system-management functions. The former is referred to as **executive monitor**, the latter **SMM-transfer monitor (STM)**.¹

The default treatment is described in Section 34.14, “Default Treatment of SMIs and SMM with VMX Operation and SMX Operation.” Dual-monitor treatment of SMM is described in Section 34.15, “Dual-Monitor Treatment of SMIs and SMM.”

34.2 SYSTEM MANAGEMENT INTERRUPT (SMI)

The only way to enter SMM is by signaling an SMI through the SMI# pin on the processor or through an SMI message received through the APIC bus. The SMI is a nonmaskable external interrupt that operates independently from the processor’s interrupt- and exception-handling mechanism and the local APIC. The SMI takes precedence over an NMI and a maskable interrupt. SMM is non-reentrant; that is, the SMI is disabled while the processor is in SMM.

NOTES

In the Pentium 4, Intel Xeon, and P6 family processors, when a processor that is designated as an application processor during an MP initialization sequence is waiting for a startup IPI (SIPI), it is in a mode where SMIs are masked. However if a SMI is received while an application processor is in the wait for SIPI mode, the SMI will be pended. The processor then responds on receipt of a SIPI by immediately servicing the pended SMI and going into SMM before handling the SIPI.

An SMI may be blocked for one instruction following execution of STI, MOV to SS, or POP into SS.

34.3 SWITCHING BETWEEN SMM AND THE OTHER PROCESSOR OPERATING MODES

Figure 2-3 shows how the processor moves between SMM and the other processor operating modes (protected, real-address, and virtual-8086). Signaling an SMI while the processor is in real-address, protected, or virtual-8086 modes always causes the processor to switch to SMM. Upon execution of the RSM instruction, the processor always returns to the mode it was in when the SMI occurred.

34.3.1 Entering SMM

The processor always handles an SMI on an architecturally defined “interruptible” point in program execution (which is commonly at an IA-32 architecture instruction boundary). When the processor receives an SMI, it waits for all instructions to retire and for all stores to complete. The processor then saves its current context in SMRAM (see Section 34.4), enters SMM, and begins to execute the SMI handler.

1. The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether it is supported.

Upon entering SMM, the processor signals external hardware that SMI handling has begun. The signaling mechanism used is implementation dependent. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is asserted each time a bus transaction is generated while the processor is in SMM. For the Pentium and Intel486 processors, the SMIACK# pin is asserted.

An SMI has a greater priority than debug exceptions and external interrupts. Thus, if an NMI, maskable hardware interrupt, or a debug exception occurs at an instruction boundary along with an SMI, only the SMI is handled. Subsequent SMI requests are not acknowledged while the processor is in SMM. The first SMI interrupt request that occurs while the processor is in SMM (that is, after SMM has been acknowledged to external hardware) is latched and serviced when the processor exits SMM with the RSM instruction. The processor will latch only one SMI while in SMM.

See Section 34.5 for a detailed description of the execution environment when in SMM.

34.3.2 Exiting From SMM

The only way to exit SMM is to execute the RSM instruction. The RSM instruction is only available to the SMI handler; if the processor is not in SMM, attempts to execute the RSM instruction result in an invalid-opcode exception (#UD) being generated.

The RSM instruction restores the processor's context by loading the state save image from SMRAM back into the processor's registers. The processor then returns an SMIACK transaction on the system bus and returns program control back to the interrupted program.

NOTES

On processors that support the shadow-stack feature, RSM loads the SSP register from the state save image in SMRAM (see Table 34-3). The value is made canonical by sign-extension before loading it into SSP.

On processors that support Intel® SGX, RSM consults the "incident to enclave mode" bit in SMRAM (see Table 34-3). If it is set, [FRED event delivery of or](#) a subsequent VM exit due to an event pending following RSM will indicate that it was incident to enclave mode. See [Section 8.3.2.1](#) and [Section 30.2.1](#) for details.

Upon successful completion of the RSM instruction, the processor signals external hardware that SMM has been exited. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is no longer generated on bus cycles. For the Pentium and Intel486 processors, the SMIACK# pin is deserted.

If the processor detects invalid state information saved in the SMRAM, it enters the shutdown state and generates a special bus cycle to indicate it has entered shutdown state. Shutdown happens only in the following situations:

- A reserved bit in control register CR4 is set to 1 on a write to CR4. This error should not happen unless SMI handler code modifies reserved areas of the SMRAM saved state map (see Section 34.4.1). CR4 is saved in the state map in a reserved location and cannot be read or modified in its saved state.
- An illegal combination of bits is written to control register CR0, in particular PG set to 1 and PE set to 0, or NW set to 1 and CD set to 0.
- CR4.PCIDE would be set to 1 and IA32_EFER.LMA to 0.
- (For the Pentium and Intel486 processors only.) If the address stored in the SMBASE register when an RSM instruction is executed is not aligned on a 32-KByte boundary. This restriction does not apply to the P6 family processors.
- CR4.CET would be set to 1 and CR0.WP to 0.
- [CR4.FRED would be set to 1 and any of the following apply:](#)
 - [IA32_EFER.LMA would be 0.](#)
 - [CPL would be 1 or 2.](#)
 - [CPL would be 0, and the logical processor would be in compatibility mode.](#)

- CPL would be 3, and either IOPL would be non-zero.

In the shutdown state, Intel processors stop executing instructions until a RESET#, INIT# or NMI# is asserted. While Pentium family processors recognize the SMI# signal in shutdown state, P6 family and Intel486 processors do not. Intel does not support using SMI# to recover from shutdown states for any processor family; the response of processors in this circumstance is not well defined. On Pentium 4 and later processors, shutdown will inhibit INTR and A20M but will not change any of the other inhibits. On these processors, NMIs will be inhibited if no action is taken in the SMI handler to uninhibit them (see Section 34.8).

If the processor is in the HALT state when the SMI is received, the processor handles the return from SMM slightly differently (see Section 34.10). Also, the SMBASE address can be changed on a return from SMM (see Section 34.11).

34.4 SMRAM

Upon entering SMM, the processor switches to a new address space. Because paging is disabled upon entering SMM, this initial address space maps all memory accesses to the low 4 GBytes of the processor's physical address space. The SMI handler's critical code and data reside in a memory region referred to as system-management RAM (SMRAM). The processor uses a pre-defined region within SMRAM to save the processor's pre-SMI context. SMRAM can also be used to store system management information (such as the system configuration and specific information about powered-down devices) and OEM-specific information.

The default SMRAM size is 64 KBytes beginning at a base physical address in physical memory called the SMBASE (see Figure 34-1). The SMBASE default value following a hardware reset is 30000H. The processor looks for the first instruction of the SMI handler at the address [SMBASE + 8000H]. It stores the processor's state in the area from [SMBASE + FE00H] to [SMBASE + FFFFH]. See Section 34.4.1 for a description of the mapping of the state save area.

The system logic is minimally required to decode the physical address range for the SMRAM from [SMBASE + 8000H] to [SMBASE + FFFFH]. A larger area can be decoded if needed. The size of this SMRAM can be between 32 KBytes and 4 GBytes.

The location of the SMRAM can be changed by changing the SMBASE value (see Section 34.11). It should be noted that all processors in a multiple-processor system are initialized with the same SMBASE value (30000H). Initialization software must sequentially place each processor in SMM and change its SMBASE so that it does not overlap those of other processors.

The actual physical location of the SMRAM can be in system memory or in a separate RAM memory. The processor generates an SMI acknowledge transaction (P6 family processors) or asserts the SMIACK# pin (Pentium and Intel486 processors) when the processor receives an SMI (see Section 34.3.1).

System logic can use the SMI acknowledge transaction or the assertion of the SMIACK# pin to decode accesses to the SMRAM and redirect them (if desired) to specific SMRAM memory. If a separate RAM memory is used for SMRAM, system logic should provide a programmable method of mapping the SMRAM into system memory space when the processor is not in SMM. This mechanism will enable start-up procedures to initialize the SMRAM space (that is, load the SMI handler) before executing the SMI handler during SMM.

34.4.1 SMRAM State Save Map

When an IA-32 processor that does not support Intel 64 architecture initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area begins at [SMBASE + 8000H + 7FFFH] and extends down to [SMBASE + 8000H + 7E00H]. Table 34-1 shows the state save map. The offset in column 1 is relative to the SMBASE value plus 8000H. Reserved spaces should not be used by software.

Some of the registers in the SMRAM state save area (marked YES in column 3) may be read and changed by the SMI handler, with the changed values restored to the processor registers by the RSM instruction. Some register images are read-only, and must not be modified (modifying these registers will result in unpredictable behavior). An SMI handler should not rely on any values stored in an area that is marked as reserved.

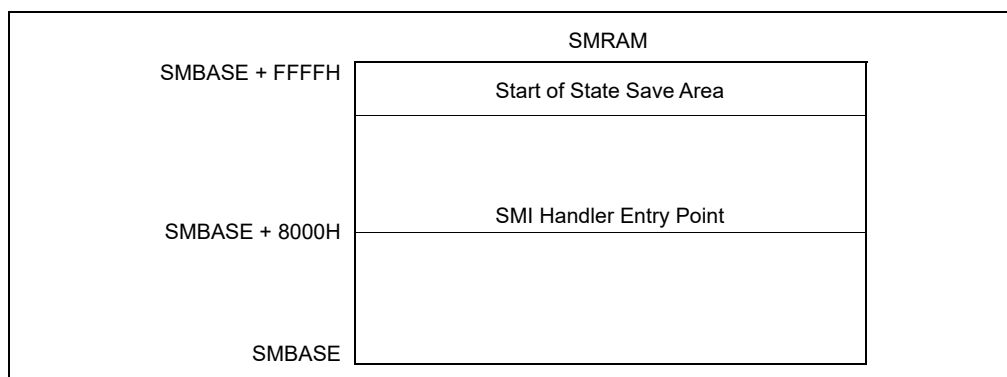


Figure 34-1. SMRAM Usage

Table 34-1. SMRAM State Save Map

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FFCH	CR0	No
7FF8H	CR3	No
7FF4H	EFLAGS	Yes
7FF0H	EIP	Yes
7FECH	EDI	Yes
7FE8H	ESI	Yes
7FE4H	EBP	Yes
7FE0H	ESP	Yes
7FDCH	EBX	Yes
7FD8H	EDX	Yes
7FD4H	ECX	Yes
7FD0H	EAX	Yes
7FCCH	DR6	No
7FC8H	DR7	No
7FC4H	TR ¹	No
7FC0H	Reserved	No
7FBCH	GS ¹	No
7FB8H	FS ¹	No
7FB4H	DS ¹	No
7FB0H	SS ¹	No
7FACH	CS ¹	No
7FA8H	ES ¹	No
7FA4H	I/O State Field, see Section 34.7	No
7FA0H	I/O Memory Address Field, see Section 34.7	No
7F9FH-7F03H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes

Table 34-1. SMRAM State Save Map (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7E00H	Reserved	No

NOTE:

1. The two most significant bytes are reserved.

The following registers are saved (but not readable) and restored upon exiting SMM:

- Control register CR4. (This register is cleared to all 0s when entering SMM).
- The hidden segment descriptor information stored in segment registers CS, DS, ES, FS, GS, and SS.

If an SMI request is issued for the purpose of powering down the processor, the values of all reserved locations in the SMM state save must be saved to nonvolatile memory.

The following state is not automatically saved and restored following an SMI and the RSM instruction, respectively:

- Debug registers DR0 through DR3.
- The x87 FPU registers.
- The MTRRs.
- Control register CR2.
- The model-specific registers (for the P6 family and Pentium processors) or test registers TR3 through TR7 (for the Pentium and Intel486 processors).
- The state of the trap controller.
- The machine-check architecture registers.
- The APIC internal interrupt state (ISR, IRR, etc.).
- The microcode update state.

If an SMI is used to power down the processor, a power-on reset will be required before returning to SMM, which will reset much of this state back to its default values. So an SMI handler that is going to trigger power down should first read these registers listed above directly, and save them (along with the rest of RAM) to nonvolatile storage. After the power-on reset, the continuation of the SMI handler should restore these values, along with the rest of the system's state. Anytime the SMI handler changes these registers in the processor, it must also save and restore them.

NOTES

A small subset of the MSRs (such as, the time-stamp counter and performance-monitoring counters) are not arbitrarily writable and therefore cannot be saved and restored. SMM-based power-down and restoration should only be performed with operating systems that do not use or rely on the values of these registers.

Operating system developers should be aware of this fact and ensure that their operating-system assisted power-down and restoration software is immune to unexpected changes in these register values.

34.4.1.1 SMRAM State Save Map and Intel 64 Architecture

When the processor initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area on an Intel 64 processor at [SMBASE + 8000H + 7FFFH] and extends to [SMBASE + 8000H + 7C00H].

Support for Intel 64 architecture is reported by CPUID.80000001:EDX[29] = 1. The layout of the SMRAM state save map is shown in Table 34-3.

Additionally, the SMRAM state save map shown in Table 34-3 also applies to processors with the following CPUID signatures listed in Table 34-2, irrespective of the value in CPUID.80000001H:EDX[29].

Table 34-2. Processor Signatures and 64-bit SMRAM State Save Map Format

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_17H	Intel Xeon Processor 5200, 5400 series, Intel Core 2 Quad processor Q9xxx, Intel Core 2 Duo processors E8000, T9000,
06_0FH	Intel Xeon Processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad, Intel Core 2 Extreme, Intel Core 2 Duo processors, Intel Pentium dual-core processors
06_1CH	45 nm Intel Atom® processors

Table 34-3. SMRAM State Save Map for Intel 64 Architecture

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FF8H	CR0	No
7FF0H	CR3	No
7FE8H	RFLAGS	Yes
7FE0H	IA32_EFER	Yes
7FD8H	RIP	Yes
7FD0H	DR6	No
7FC8H	DR7	No
7FC4H	TR SEL ¹	No
7FC0H	LDTR SEL ¹	No
7FBCH	GS SEL ¹	No
7FB8H	FS SEL ¹	No
7FB4H	DS SEL ¹	No
7FB0H	SS SEL ¹	No
7FACH	CS SEL ¹	No
7FA8H	ES SEL ¹	No
7FA4H	IO_MISC	No
7F9CH	IO_MEM_ADDR	No
7F94H	RDI	Yes
7F8CH	RSI	Yes
7F84H	RBP	Yes
7F7CH	RSP	Yes
7F74H	RBX	Yes
7F6CH	RDX	Yes
7F64H	RCX	Yes
7F5CH	RAX	Yes
7F54H	R8	Yes
7F4CH	R9	Yes

Table 34-3. SMRAM State Save Map for Intel 64 Architecture (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7F44H	R10	Yes
7F3CH	R11	Yes
7F34H	R12	Yes
7F2CH	R13	Yes
7F24H	R14	Yes
7F1CH	R15	Yes
7F1BH-7F04H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7EE4H	Reserved	No
7EE0H	Bit 0 is set if the SMI was incident to VMX non-root operation and the “enable EPT” VM-execution control was 1. Bit 1 is set if the SMI was incident to enclave mode.	No
7ED8H	Value of EPTP VM-execution control field	No
7ED7H - 7ECC0H	Reserved	No
7EC8H	SSP	Yes
7EC7H - 7EA0H	Reserved	No
7E9CH	LDT Base (lower 32 bits)	No
7E98H	Reserved	No
7E94H	IDT Base (lower 32 bits)	No
7E90H	Reserved	No
7E8CH	GDT Base (lower 32 bits)	No
7E8BH - 7E48H	Reserved	No
7E40H	CR4 (64 bits)	No
7E3FH - 7DF0H	Reserved	No
7DE8H	IO_RIP	Yes
7DE7H - 7DDCH	Reserved	No
7DD8H	IDT Base (Upper 32 bits)	No
7DD4H	LDT Base (Upper 32 bits)	No
7DD0H	GDT Base (Upper 32 bits)	No
7DCFH - 7C00H	Reserved	No

NOTE:

1. The two most significant bytes are reserved.

34.4.2 SMRAM Caching

An IA-32 processor does not automatically write back and invalidate its caches before entering SMM or before exiting SMM. Because of this behavior, care must be taken in the placement of the SMRAM in system memory and

in the caching of the SMRAM to prevent cache incoherence when switching back and forth between SMM and protected mode operation. Any of the following three methods of locating the SMRAM in system memory will guarantee cache coherency.

- Place the SMRAM in a dedicated section of system memory that the operating system and applications are prevented from accessing. Here, the SMRAM can be designated as cacheable (WB, WT, or WC) for optimum processor performance, without risking cache incoherence when entering or exiting SMM.
- Place the SMRAM in a section of memory that overlaps an area used by the operating system (such as the video memory), but designate the SMRAM as uncacheable (UC). This method prevents cache access when in SMM to maintain cache coherency, but the use of uncacheable memory reduces the performance of SMM code.
- Place the SMRAM in a section of system memory that overlaps an area used by the operating system and/or application code, but explicitly flush (write back and invalidate) the caches upon entering and exiting SMM mode. This method maintains cache coherency, but incurs the overhead of two complete cache flushes.

For Pentium 4, Intel Xeon, and P6 family processors, a combination of the first two methods of locating the SMRAM is recommended. Here the SMRAM is split between an overlapping and a dedicated region of memory. Upon entering SMM, the SMRAM space that is accessed overlaps video memory (typically located in low memory). This SMRAM section is designated as UC memory. The initial SMM code then jumps to a second SMRAM section that is located in a dedicated region of system memory (typically in high memory). This SMRAM section can be cached for optimum processor performance.

For systems that explicitly flush the caches upon entering SMM (the third method described above), the cache flush can be accomplished by asserting the FLUSH# pin at the same time as the request to enter SMM (generally initiated by asserting the SMI# pin). The priorities of the FLUSH# and SMI# pins are such that the FLUSH# is serviced first. To guarantee this behavior, the processor requires that the following constraints on the interaction of FLUSH# and SMI# be met. In a system where the FLUSH# and SMI# pins are synchronous and the set up and hold times are met, then the FLUSH# and SMI# pins may be asserted in the same clock. In asynchronous systems, the FLUSH# pin must be asserted at least one clock before the SMI# pin to guarantee that the FLUSH# pin is serviced first.

Upon leaving SMM (for systems that explicitly flush the caches), the WBINVD instruction should be executed prior to leaving SMM to flush the caches.

NOTES

In systems based on the Pentium processor that use the FLUSH# pin to write back and invalidate cache contents before entering SMM, the processor will prefetch at least one cache line in between when the Flush Acknowledge cycle is run and the subsequent recognition of SMI# and the assertion of SMIACK#.

It is the obligation of the system to ensure that these lines are not cached by returning KEN# inactive to the Pentium processor.

34.4.2.1 System Management Range Registers (SMRR)

SMI handler code and data stored by SMM code resides in SMRAM. The SMRR interface is an enhancement in Intel 64 architecture to limit cacheable reference of addresses in SMRAM to code running in SMM. The SMRR interface can be configured only by code running in SMM. Details of SMRR is described in Section 14.11.2.4.

34.5 SMI HANDLER EXECUTION ENVIRONMENT

Section 34.5.1 describes the initial execution environment for an SMI handler. An SMI handler may re-configure its execution environment to other supported operating modes. Section 34.5.2 discusses modifications an SMI handler can make to its execution environment. Section 34.5.3 discusses Control-flow Enforcement Technology (CET) interactions in the environment.

34.5.1 Initial SMM Execution Environment

After saving the current context of the processor, the processor initializes its core registers to the values shown in Table 34-4. Upon entering SMM, the PE and PG flags in control register CR0 are cleared, which places the processor in an environment similar to real-address mode. The differences between the SMM execution environment and the real-address mode execution environment are as follows:

- The addressable address space ranges from 0 to FFFFFFFFH (4 GBytes).
- The normal 64-KByte segment limit for real-address mode is increased to 4 GBytes.
- The default operand and address sizes are set to 16 bits, which restricts the addressable SMRAM address space to the 1-MByte real-address mode limit for native real-address-mode code. However, operand-size and address-size override prefixes can be used to access the address space beyond the 1-MByte.

Table 34-4. Processor Register Initialization in SMM

Register	Contents
General-purpose registers	Undefined
EFLAGS	00000002H
EIP	00008000H
CS selector	SMM Base shifted right 4 bits (default 3000H)
CS base	SMM Base (default 30000H)
DS, ES, FS, GS, SS Selectors	0000H
DS, ES, FS, GS, SS Bases	00000000H
DS, ES, FS, GS, SS Limits	0FFFFFFFH
CR0	PE, EM, TS, and PG flags set to 0; others unmodified
CR4	Cleared to zero
DR6	Undefined
DR7	00000400H

- Near jumps and calls can be made to anywhere in the 4-GByte address space if a 32-bit operand-size override prefix is used. Due to the real-address-mode style of base-address formation, a far call or jump cannot transfer control to a segment with a base address of more than 20 bits (1 MByte). However, since the segment limit in SMM is 4 GBytes, offsets into a segment that go beyond the 1-MByte limit are allowed when using 32-bit operand-size override prefixes. Any program control transfer that does not have a 32-bit operand-size override prefix truncates the EIP value to the 16 low-order bits.
- Data and the stack can be located anywhere in the 4-GByte address space, but can be accessed only with a 32-bit address-size override if they are located above 1 MByte. As with the code segment, the base address for a data or stack segment cannot be more than 20 bits.

The value in segment register CS is automatically set to the default of 30000H for the SMBASE shifted 4 bits to the right; that is, 3000H. The EIP register is set to 8000H. When the EIP value is added to shifted CS value (the SMBASE), the resulting linear address points to the first instruction of the SMI handler.

The other segment registers (DS, SS, ES, FS, and GS) are cleared to 0 and their segment limits are set to 4 GBytes. In this state, the SMRAM address space may be treated as a single flat 4-GByte linear address space. If a segment register is loaded with a 16-bit value, that value is then shifted left by 4 bits and loaded into the segment base (hidden part of the segment register). The limits and attributes are not modified.

Maskable hardware interrupts, exceptions, NMI interrupts, SMI interrupts, A20M interrupts, single-step traps, breakpoint traps, and INIT operations are inhibited when the processor enters SMM. Maskable hardware interrupts, exceptions, single-step traps, and breakpoint traps can be enabled in SMM if the SMM execution environment provides and initializes an interrupt table and the necessary interrupt and exception handlers (see Section 34.6).

34.5.2 SMI Handler Operating Mode Switching

Within SMM, an SMI handler may change the processor's operating mode (e.g., to enable PAE paging, enter 64-bit mode, etc.) after it has made proper preparation and initialization to do so. For example, if switching to 32-bit protected mode, the SMI handler should follow the guidelines provided in Chapter 12, "Processor Management and Initialization." If the SMI handler does wish to change operating mode, it is responsible for executing the appropriate mode-transition code after each SMI.

It is recommended that the SMI handler make use of all means available to protect the integrity of its critical code and data. In particular, it should use the system-management range register (SMRR) interface if it is available (see Section 11.11.2.4). The SMRR interface can protect only the first 4 GBytes of the physical address space. The SMI handler should take that fact into account if it uses operating modes that allow access to physical addresses beyond that 4-GByte limit (e.g., PAE paging or 64-bit mode).

Execution of the RSM instruction restores the pre-SMI processor state from the SMRAM state-state map (see Section 34.4.1) into which it was stored when the processor entered SMM. (The SMBASE field in the SMRAM state-save map does not determine the state following RSM but rather the initial environment following the next entry to SMM.) Any required change to operating mode is performed by the RSM instruction; there is no need for the SMI handler to change modes explicitly prior to executing RSM.

34.5.3 Control-flow Enforcement Technology Interactions

On processors that support CET shadow stacks, when the processor enters SMM, the processor saves the SSP register to the SMRAM state save area (see Table 34-3) and clears CR4.CET to 0. Thus, the initial execution environment of the SMI handler has CET disabled and all of the CET state of the interrupted program is still in the machine. An SMM that uses CET is required to save the interrupted program's CET state and restore the CET state prior to exiting SMM.

34.6 EXCEPTIONS AND INTERRUPTS WITHIN SMM

When the processor enters SMM, all hardware interrupts are disabled in the following manner:

- The IF flag in the EFLAGS register is cleared, which inhibits maskable hardware interrupts from being generated.
- The TF flag in the EFLAGS register is cleared, which disables single-step traps.
- Debug register DR7 is cleared, which disables breakpoint traps. (This action prevents a debugger from accidentally breaking into an SMI handler if a debug breakpoint is set in normal address space that overlays code or data in SMRAM.)
- NMI, SMI, and A20M interrupts are blocked by internal SMM logic. (See Section 34.8 for more information about how NMIs are handled in SMM.)

Software-invoked interrupts and exceptions can still occur, and maskable hardware interrupts can be enabled by setting the IF flag. Intel recommends that SMM code be written in so that it does not invoke software interrupts (with the INT *n*, INTO, INT1, INT3, or BOUND instructions) or generate exceptions.

If the SMI handler requires interrupt and exception handling, an SMM interrupt table and the necessary exception and interrupt handlers must be created and initialized from within SMM. Until the interrupt table is correctly initialized (using the LIDT instruction), exceptions and software interrupts will result in unpredictable processor behavior.

The following restrictions apply when designing SMM interrupt and exception-handling facilities:

- The interrupt table should be located at linear address 0 and must contain real-address mode style interrupt vectors (4 bytes containing CS and IP).
- Due to the real-address mode style of base address formation, an interrupt or exception cannot transfer control to a segment with a base address of more than 20 bits.
- An interrupt or exception cannot transfer control to a segment offset of more than 16 bits (64 KBytes).

- When an exception or interrupt occurs, only the 16 least-significant bits of the return address (EIP) are pushed onto the stack. If the offset of the interrupted procedure is greater than 64 KBytes, it is not possible for the interrupt/exception handler to return control to that procedure. (One solution to this problem is for a handler to adjust the return address on the stack.)
- The SMBASE relocation feature affects the way the processor will return from an interrupt or exception generated while the SMI handler is executing. For example, if the SMBASE is relocated to above 1 MByte, but the exception handlers are below 1 MByte, a normal return to the SMI handler is not possible. One solution is to provide the exception handler with a mechanism for calculating a return address above 1 MByte from the 16-bit return address on the stack, then use a 32-bit far call to return to the interrupted procedure.
- If an SMI handler needs access to the debug trap facilities, it must ensure that an SMM accessible debug handler is available and save the current contents of debug registers DR0 through DR3 (for later restoration). Debug registers DR0 through DR3 and DR7 must then be initialized with the appropriate values.
- If an SMI handler needs access to the single-step mechanism, it must ensure that an SMM accessible single-step handler is available, and then set the TF flag in the EFLAGS register.
- If the SMI design requires the processor to respond to maskable hardware interrupts or software-generated interrupts while in SMM, it must ensure that SMM accessible interrupt handlers are available and then set the IF flag in the EFLAGS register (using the STI instruction). Software interrupts are not blocked upon entry to SMM, so they do not need to be enabled.

34.7 MANAGING SYNCHRONOUS AND ASYNCHRONOUS SYSTEM MANAGEMENT INTERRUPTS

When coding for a multiprocessor system or a system with Intel HT Technology, it was not always possible for an SMI handler to distinguish between a synchronous SMI (triggered during an I/O instruction) and an asynchronous SMI. To facilitate the discrimination of these two events, incremental state information has been added to the SMM state save map.

Processors that have an SMM revision ID of 30004H or higher have the incremental state information described below.

34.7.1 I/O State Implementation

Within the extended SMM state save map, a bit (IO_SMI) is provided that is set only when an SMI is either taken immediately after a *successful* I/O instruction or is taken after a *successful* iteration of a REP I/O instruction (the *successful* notion pertains to the processor point of view; not necessarily to the corresponding platform function). When set, the IO_SMI bit provides a strong indication that the corresponding SMI was synchronous. In this case, the SMM State Save Map also supplies the port address of the I/O operation. The IO_SMI bit and the I/O Port Address may be used in conjunction with the information logged by the platform to confirm that the SMI was indeed synchronous.

The IO_SMI bit by itself is a strong indication, not a guarantee, that the SMI is synchronous. This is because an asynchronous SMI might coincidentally be taken after an I/O instruction. In such a case, the IO_SMI bit would still be set in the SMM state save map.

Information characterizing the I/O instruction is saved in two locations in the SMM State Save Map (Table 34-5). The IO_SMI bit also serves as a valid bit for the rest of the I/O information fields. The contents of these I/O information fields are not defined when the IO_SMI bit is not set.

Table 34-5. I/O Instruction Information in the SMM State Save Map

State (SMM Rev. ID: 30004H or higher)	Format									
	31	16	15	8	7	4	3	1	0	

Table 34-5. I/O Instruction Information in the SMM State Save Map

I/O State Field SMRAM offset 7FA4	I/O Port	Reserved	I/O Type	I/O Length	IO_SMI
	31				0
I/O Memory Address Field SMRAM offset 7FA0	I/O Memory Address				

When IO_SMI is set, the other fields may be interpreted as follows:

- I/O length:
 - 001 – Byte
 - 010 – Word
 - 100 – Dword
- I/O instruction type (Table 34-6)

Table 34-6. I/O Instruction Type Encodings

Instruction	Encoding
IN Immediate	1001
IN DX	0001
OUT Immediate	1000
OUT DX	0000
INS	0011
OUTS	0010
REP INS	0111
REP OUTS	0110

34.8 NMI HANDLING WHILE IN SMM

NMI interrupts are blocked upon entry to the SMI handler. If an NMI request occurs during the SMI handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMI handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequence. This assumes that NMIs were not blocked before the SMI occurred. If NMIs were blocked before the SMI occurred, they are blocked after execution of RSM.

Although NMI requests are blocked when the processor enters SMM, they may be enabled through software by executing an IRET instruction. If the SMI handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine for the purpose of executing an IRET instruction. Once an IRET instruction is executed, NMI interrupt requests are serviced in the same “real mode” manner in which they are handled outside of SMM.

Also, for the Pentium processor, exceptions that invoke a trap or fault handler will enable NMI interrupts from inside of SMM. This behavior is implementation specific for the Pentium processor and is not part of the IA-32 architecture.

34.9 SMM REVISION IDENTIFIER

The SMM revision identifier field is used to indicate the version of SMM and the SMM extensions that are supported by the processor (see Figure 34-2). The SMM revision identifier is written during SMM entry and can be examined in SMRAM space at offset 7EFCH. The lower word of the SMM revision identifier refers to the version of the base SMM architecture.

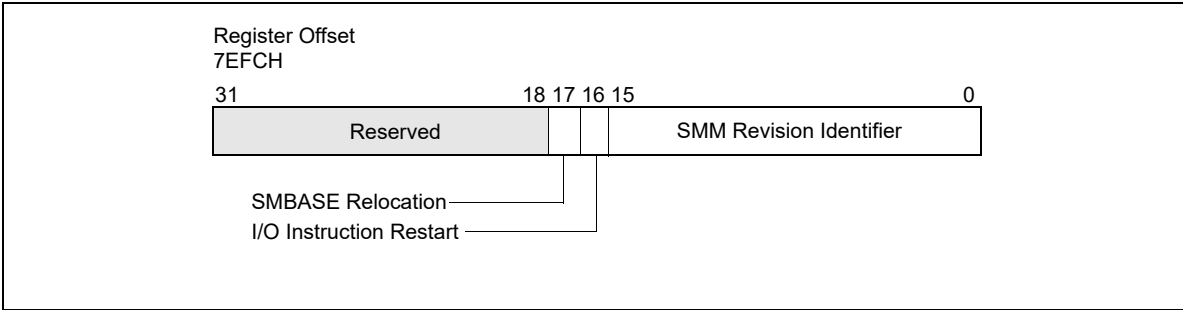


Figure 34-2. SMM Revision Identifier

The upper word of the SMM revision identifier refers to the extensions available. If the I/O instruction restart flag (bit 16) is set, the processor supports the I/O instruction restart (see Section 34.12); if the SMBASE relocation flag (bit 17) is set, SMRAM base address relocation is supported (see Section 34.11).

34.10 AUTO HALT RESTART

If the processor is in a HALT state (due to the prior execution of a HLT instruction) when it receives an SMI, the processor records the fact in the auto HALT restart flag in the saved processor state (see Figure 34-3). (This flag is located at offset 7F02H and bit 0 in the state save area of the SMRAM.)

If the processor sets the auto HALT restart flag upon entering SMM (indicating that the SMI occurred when the processor was in the HALT state), the SMI handler has two options:

- It can leave the auto HALT restart flag set, which instructs the RSM instruction to return program control to the HLT instruction. This option in effect causes the processor to re-enter the HALT state after handling the SMI. (This is the default operation.)
- It can clear the auto HALT restart flag, which instructs the RSM instruction to return program control to the instruction following the HLT instruction.

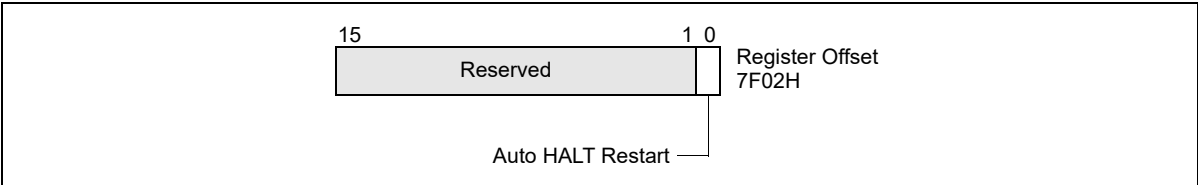


Figure 34-3. Auto HALT Restart Field

These options are summarized in Table 34-7. If the processor was not in a HALT state when the SMI was received (the auto HALT restart flag is cleared), setting the flag to 1 will cause unpredictable behavior when the RSM instruction is executed.

Table 34-7. Auto HALT Restart Flag Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
0	0	Returns to next instruction in interrupted program or task.
0	1	Unpredictable.
1	0	Returns to next instruction after HLT instruction.
1	1	Returns to HALT state.

If the HLT instruction is restarted, the processor will generate a memory access to fetch the HLT instruction (if it is

not in the internal cache), and execute a HLT bus transaction. This behavior results in multiple HLT bus transactions for the same HLT instruction.

34.10.1 Executing the HLT Instruction in SMM

The HLT instruction should not be executed during SMM, unless interrupts have been enabled by setting the IF flag in the EFLAGS register. If the processor is halted in SMM, the only event that can remove the processor from this state is a maskable hardware interrupt or a hardware reset.

34.11 SMBASE RELOCATION

The default base address for the SMRAM is 30000H. This value is contained in an internal processor register called the SMBASE register. Software can relocate the SMRAM by setting the SMBASE field in the saved state map (at offset 7EF8H) to a new value (see Figure 34-4). The RSM instruction reloads the internal SMBASE register with the value in the SMBASE field each time it exits SMM. All subsequent SMI requests will use the new SMBASE value to find the starting address for the SMI handler (at SMBASE + 8000H) and the SMRAM state save area (from SMBASE + FE00H to SMBASE + FFFFH). (The processor resets the value in its internal SMBASE register to 30000H on a RESET, but does not change it on an INIT.)

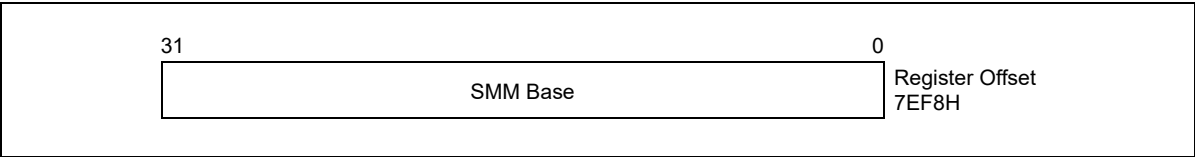


Figure 34-4. SMBASE Relocation Field

In multiple-processor systems, initialization software must adjust the SMBASE value for each processor so that the SMRAM state save areas for each processor do not overlap. (For Pentium and Intel486 processors, the SMBASE values must be aligned on a 32-KByte boundary or the processor will enter shutdown state during the execution of a RSM instruction.)

If the SMBASE relocation flag in the SMM revision identifier field is set, it indicates the ability to relocate the SMBASE (see Section 34.9).

34.12 I/O INSTRUCTION RESTART

If the I/O instruction restart flag in the SMM revision identifier field is set (see Section 34.9), the I/O instruction restart mechanism is present on the processor. This mechanism allows an interrupted I/O instruction to be re-executed upon returning from SMM mode. For example, if an I/O instruction is used to access a powered-down I/O device, a chipset supporting this device can intercept the access and respond by asserting SMI#. This action invokes the SMI handler to power-up the device. Upon returning from the SMI handler, the I/O instruction restart mechanism can be used to re-execute the I/O instruction that caused the SMI.

The I/O instruction restart field (at offset 7F00H in the SMM state-save area, see Figure 34-5) controls I/O instruction restart. When an RSM instruction is executed, if this field contains the value FFH, then the EIP register is modified to point to the I/O instruction that received the SMI request. The processor will then automatically re-execute the I/O instruction that the SMI trapped. (The processor saves the necessary machine state to ensure that re-execution of the instruction is handled coherently.)



Figure 34-5. I/O Instruction Restart Field

If the I/O instruction restart field contains the value 00H when the RSM instruction is executed, then the processor begins program execution with the instruction following the I/O instruction. (When a repeat prefix is being used, the next instruction may be the next I/O instruction in the repeat loop.) Not re-executing the interrupted I/O instruction is the default behavior; the processor automatically initializes the I/O instruction restart field to 00H upon entering SMM. Table 34-8 summarizes the states of the I/O instruction restart field.

Table 34-8. I/O Instruction Restart Field Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
00H	00H	Does not re-execute trapped I/O instruction.
00H	FFH	Re-executes trapped I/O instruction.

The I/O instruction restart mechanism does not indicate the cause of the SMI. It is the responsibility of the SMI handler to examine the state of the processor to determine the cause of the SMI and to determine if an I/O instruction was interrupted and should be restarted upon exiting SMM. If an SMI interrupt is signaled on a non-I/O instruction boundary, setting the I/O instruction restart field to FFH prior to executing the RSM instruction will likely result in a program error.

34.12.1 Back-to-Back SMI Interrupts When I/O Instruction Restart Is Being Used

If an SMI interrupt is signaled while the processor is servicing an SMI interrupt that occurred on an I/O instruction boundary, the processor will service the new SMI request before restarting the originally interrupted I/O instruction. If the I/O instruction restart field is set to FFH prior to returning from the second SMI handler, the EIP will point to an address different from the originally interrupted I/O instruction, which will likely lead to a program error. To avoid this situation, the SMI handler must be able to recognize the occurrence of back-to-back SMI interrupts when I/O instruction restart is being used and ensure that the handler sets the I/O instruction restart field to 00H prior to returning from the second invocation of the SMI handler.

34.13 SMM MULTIPLE-PROCESSOR CONSIDERATIONS

The following should be noted when designing multiple-processor systems:

- Any processor in a multiprocessor system can respond to an SMI.
- Each processor needs its own SMRAM space. This space can be in system memory or in a separate RAM.
- The SMRAMs for different processors can be overlapped in the same memory space. The only stipulation is that each processor needs its own state save area and its own dynamic data storage area. (Also, for the Pentium and Intel486 processors, the SMBASE address must be located on a 32-KByte boundary.) Code and static data can be shared among processors. Overlapping SMRAM spaces can be done more efficiently with the P6 family processors because they do not require that the SMBASE address be on a 32-KByte boundary.
- The SMI handler will need to initialize the SMBASE for each processor.
- Processors can respond to local SMIs through their SMI# pins or to SMIs received through the APIC interface. The APIC interface can distribute SMIs to different processors.
- Two or more processors can be executing in SMM at the same time.

- When operating Pentium processors in dual processing (DP) mode, the SMI $\overline{ACT\#}$ pin is driven only by the MRM processor and should be sampled with ADS $\#$. For additional details, see Chapter 14 of the Pentium Processor Family User's Manual, Volume 1.

SMM is not re-entrant, because the SMRAM State Save Map is fixed relative to the SMBASE. If there is a need to support two or more processors in SMM mode at the same time then each processor should have dedicated SMRAM spaces. This can be done by using the SMBASE Relocation feature (see Section 34.11).

34.14 DEFAULT TREATMENT OF SMIS AND SMM WITH VMX OPERATION AND SMX OPERATION

Under the default treatment, the interactions of SMIs and SMM with VMX operation are few. This section details those interactions. It also explains how this treatment affects SMX operation.

34.14.1 Default Treatment of SMI Delivery

Ordinary SMI delivery saves processor state into SMRAM and then loads state based on architectural definitions. Under the default treatment, processors that support VMX operation perform SMI delivery as follows:

```

enter SMM;
save the following internal to the processor:
    CR4.VMXE
    an indication of whether the logical processor was in VMX operation (root or non-root)
IF the logical processor is in VMX operation
    THEN
        save current VMCS pointer internal to the processor;
        leave VMX operation;
        save VMX-critical state defined below;
FI;
IF the logical processor supports SMX operation
    THEN
        save internal to the logical processor an indication of whether the Intel® TXT private space is locked;
        IF the TXT private space is unlocked
            THEN lock the TXT private space;
        FI;
FI;
CR4.VMXE := 0;
perform ordinary SMI delivery:
    save processor state in SMRAM;
    set processor state to standard SMM values;1
    invalidate linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H
are invalidated for all EPT $\overline{RTA}$  values (EPT $\overline{RTA}$  is the value of bits 51:12 of EPTP; see Section 31.4);
  
```

The pseudocode above makes reference to the saving of **VMX-critical state**. This state consists of the following: (1) SS.DPL (the current privilege level); (2) RFLAGS.VM²; (3) the state of blocking by STI and by MOV SS (see Table 27-3 in Section •); (4) the state of virtual-NMI blocking (only if the processor is in VMX non-root operation and the “virtual NMIs” VM-execution control is 1); and (5) an indication of whether an MTF VM exit is pending (see Section 28.5.2). These data may be saved internal to the processor or in the VMCS region of the current VMCS.

-
- This causes the logical processor to block INIT signals, NMIs, and SMIs.
 - Section 34.14 and Section 34.15 use the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of these registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to the lower 32 bits of the register.

Processors that do not support SMI recognition while there is blocking by STI or by MOV SS need not save the state of such blocking.

If the logical processor supports the 1-setting of the “enable EPT” VM-execution control and the logical processor was in VMX non-root operation at the time of an SMI, it saves the value of that control into bit 0 of the 32-bit field at offset SMBASE + 8000H + 7EE0H (SMBASE + FEE0H; see Table 34-3).¹ If the logical processor was not in VMX non-root operation at the time of the SMI, it saves 0 into that bit. If the logical processor saves 1 into that bit (it was in VMX non-root operation and the “enable EPT” VM-execution control was 1), it saves the value of the EPT pointer (EPTP) into the 64-bit field at offset SMBASE + 8000H + 7ED8H (SMBASE + FED8H).

Because SMI delivery causes a logical processor to leave VMX operation, all the controls associated with VMX non-root operation are disabled in SMM and thus cannot cause VM exits while the logical processor in SMM.

34.14.2 Default Treatment of RSM

Ordinary execution of RSM restores processor state from SMRAM. Under the default treatment, processors that support VMX operation perform RSM as follows:

```

IF VMXE = 1 in CR4 image in SMRAM
    THEN fail and enter shutdown state;
ELSE
    restore state normally from SMRAM;
    invalidate linear mappings and combined mappings associated with all VPIDs and all PCIDs; combined mappings are invalidated
for all EPTRTA values (EPTRTA is the value of bits 51:12 of EPTP; see Section 31.4);
    IF the logical processor supports SMX operation and the Intel® TXT private space was unlocked at the time of the last SMI (as
saved)
        THEN unlock the TXT private space;
    FI;
    CR4.VMXE := value stored internally;
    IF internal storage indicates that the logical processor
    had been in VMX operation (root or non-root)
        THEN
            enter VMX operation (root or non-root);
            restore VMX-critical state as defined in Section 34.14.1;
            set to their fixed values any bits in CR0 and CR4 whose values must be fixed in VMX operation (see Section 26.8);2
            IF RFLAGS.VM = 0 AND (in VMX root operation OR the “unrestricted guest” VM-execution control is 0)3
                THEN
                    CS.RPL := SS.DPL;
                    SS.RPL := SS.DPL;
            FI;
            restore current VMCS pointer;
        FI;
    leave SMM;
    IF logical processor will be in VMX operation or in SMX operation after RSM
        THEN block A20M and leave A20M mode;
    FI;
FI;

```

1. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, SMI functions as the “enable EPT” VM-execution control were 0. See Section 27.6.2.
2. If the RSM is to VMX non-root operation and both the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls will be 1, CR0.PE and CR0.PG retain the values that were loaded from SMRAM regardless of what is reported in the capability MSR IA32_VMX_CRO_FIXED0.
3. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 27.6.2.

RSM unblocks SMIs. It restores the state of blocking by NMI (see Table 27-3 in Section •) as follows:

- If the RSM is not to VMX non-root operation or if the “virtual NMIs” VM-execution control will be 0, the state of NMI blocking is restored normally.
- If the RSM is to VMX non-root operation and the “virtual NMIs” VM-execution control will be 1, NMIs are not blocked after RSM. The state of virtual-NMI blocking is restored as part of VMX-critical state.

INIT signals are blocked after RSM if and only if the logical processor will be in VMX root operation.

If RSM returns a logical processor to VMX non-root operation, it re-establishes the controls associated with the current VMCS. If the “interrupt-window exiting” VM-execution control is 1, a VM exit occurs immediately after RSM if the enabling conditions apply. The same is true for the “NMI-window exiting” VM-execution control. Such VM exits occur with their normal priority. See Section 28.2.

If an MTF VM exit was pending at the time of the previous SMI, an MTF VM exit is pending on the instruction boundary following execution of RSM. The following items detail the treatment of MTF VM exits that may be pending following RSM:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these MTF VM exits. These MTF VM exits take priority over debug-trap exceptions and lower priority events.
- These MTF VM exits wake the logical processor if RSM caused the logical processor to enter the HLT state (see Section 34.10). They do not occur if the logical processor just entered the shutdown state.

34.14.3 Protection of CR4.VMXE in SMM

Under the default treatment, CR4.VMXE is treated as a reserved bit while a logical processor is in SMM. Any attempt by software running in SMM to set this bit causes a general-protection exception. In addition, software cannot use VMX instructions or enter VMX operation while in SMM.

34.14.4 VMXOFF and SMI Unblocking

The VMXOFF instruction can be executed only with the default treatment (see Section 34.15.1) and only outside SMM. If SMIs are blocked when VMXOFF is executed, VMXOFF unblocks them unless IA32_SMM_MONITOR_CTL[bit 2] is 1 (see Section 34.15.5 for details regarding this MSR).¹ Section 34.15.7 identifies a case in which SMIs may be blocked when VMXOFF is executed.

Not all processors allow this bit to be set to 1. Software should consult the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine whether this is allowed.

34.15 DUAL-MONITOR TREATMENT OF SMIs AND SMM

Dual-monitor treatment is activated through the cooperation of the **executive monitor** (the VMM that operates outside of SMM to provide basic virtualization) and the **SMM-transfer monitor (STM)**; the VMM that operates inside SMM—while in VMX operation—to support system-management functions). Control is transferred to the STM through VM exits; VM entries are used to return from SMM.

The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether it is supported.

34.15.1 Dual-Monitor Treatment Overview

The dual-monitor treatment uses an executive monitor and an SMM-transfer monitor (STM). Transitions from the executive monitor or its guests to the STM are called **SMM VM exits** and are discussed in Section 34.15.2. SMM

1. Setting IA32_SMM_MONITOR_CTL[bit 2] to 1 prevents VMXOFF from unblocking SMIs regardless of the value of the register’s valid bit (bit 0).

VM exits are caused by SMIs as well as executions of VMCALL in VMX root operation. The latter allow the executive monitor to call the STM for service.

The STM runs in VMX root operation and uses VMX instructions to establish a VMCS and perform VM entries to its own guests. This is done all inside SMM (see Section 34.15.3). The STM returns from SMM, not by using the RSM instruction, but by using a VM entry that returns from SMM. Such VM entries are described in Section 34.15.4.

Initially, there is no STM and the default treatment (Section 34.14) is used. The dual-monitor treatment is not used until it is enabled and activated. The steps to do this are described in Section 34.15.5 and Section 34.15.6.

It is not possible to leave VMX operation under the dual-monitor treatment; VMXOFF will fail if executed. The dual-monitor treatment must be deactivated first. The STM deactivates dual-monitor treatment using a VM entry that returns from SMM with the “deactivate dual-monitor treatment” VM-entry control set to 1 (see Section 34.15.7).

The executive monitor configures any VMCS that it uses for VM exits to the executive monitor. SMM VM exits, which transfer control to the STM, use a different VMCS. Under the dual-monitor treatment, each logical processor uses a separate VMCS called the **SMM-transfer VMCS**. When the dual-monitor treatment is active, the logical processor maintains another VMCS pointer called the **SMM-transfer VMCS pointer**. The SMM-transfer VMCS pointer is established when the dual-monitor treatment is activated.

34.15.2 SMM VM Exits

An SMM VM exit is a VM exit that begins outside SMM and that ends in SMM.

Unlike other VM exits, SMM VM exits can begin in VMX root operation. SMM VM exits result from the arrival of an SMI outside SMM or from execution of VMCALL in VMX root operation outside SMM. Execution of VMCALL in VMX root operation causes an SMM VM exit only if the valid bit is set in the IA32_SMM_MONITOR_CTL MSR (see Section 34.15.5).

Execution of VMCALL in VMX root operation causes an SMM VM exit even under the default treatment. This SMM VM exit activates the dual-monitor treatment (see Section 34.15.6).

Differences between SMM VM exits and other VM exits are detailed in Sections 34.15.2.1 through 34.15.2.5. Differences between SMM VM exits that activate the dual-monitor treatment and other SMM VM exits are described in Section 34.15.6.

34.15.2.1 Architectural State Before a VM Exit

System-management interrupts (SMIs) that cause SMM VM exits always do so directly. They do not save state to SMRAM as they do under the default treatment.

34.15.2.2 Updating the Current-VMCS and Executive-VMCS Pointers

SMM VM exits begin by performing the following steps:

1. The executive-VMCS pointer field in the SMM-transfer VMCS is loaded as follows:
 - If the SMM VM exit commenced in VMX non-root operation, it receives the current-VMCS pointer.
 - If the SMM VM exit commenced in VMX root operation, it receives the VMXON pointer.
2. The current-VMCS pointer is loaded with the value of the SMM-transfer VMCS pointer.

The last step ensures that the current VMCS is the SMM-transfer VMCS. VM-exit information is recorded in that VMCS, and VM-entry control fields in that VMCS are updated. State is saved into the guest-state area of that VMCS. The VM-exit controls and host-state area of that VMCS determine how the VM exit operates.

34.15.2.3 Recording VM-Exit Information

SMM VM exits differ from other VM exit with regard to the way they record VM-exit information. The differences follow.

- **Exit reason.**

- Bits 15:0 of this field contain the basic exit reason. The field is loaded with the reason for the SMM VM exit: I/O SMI (an SMI arrived immediately after retirement of an I/O instruction), other SMI, or VMCALL. See Appendix C, “VMX Basic Exit Reasons.”
- SMM VM exits are the only VM exits that may occur in VMX root operation. Because the SMM-transfer monitor may need to know whether it was invoked from VMX root or VMX non-root operation, this information is stored in bit 29 of the exit-reason field (see Table 27-19 in Section 27.9.1). The bit is set by SMM VM exits from VMX root operation.
- If the SMM VM exit occurred in VMX non-root operation and an MTF VM exit was pending, bit 28 of the exit-reason field is set; otherwise, it is cleared.
- Bits 27:16 and bits 31:30 are cleared.
- **Exit qualification.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, the exit qualification contains information about the I/O instruction that retired immediately before the SMI. It has the format given in Table 34-9.

Table 34-9. Exit Qualification for SMIs That Arrive Immediately After the Retirement of an I/O Instruction

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used.
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Reserved (cleared to 0)
31:16	Port number (as specified in the I/O instruction)
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

- **Guest linear address.** This field is used for VM exits due to SMIs that arrive immediately after the retirement of an INS or OUTS instruction for which the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) is usable. The field receives the value of the linear address generated by ES:(E)DI (for INS) or segment:(E)SI (for OUTS; the default segment is DS but can be overridden by a segment override prefix) at the time the instruction started. If the relevant segment is not usable, the value is undefined. On processors that support Intel 64 architecture, bits 63:32 are clear if the logical processor was not in 64-bit mode before the VM exit.
- **I/O RCX, I/O RSI, I/O RDI, and I/O RIP.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, these fields receive the values that were in RCX, RSI, RDI, and RIP, respectively, before the I/O instruction executed. Thus, the value saved for I/O RIP addresses the I/O instruction.

34.15.2.4 Saving Guest State

SMM VM exits save the contents of the SMBASE register into the corresponding field in the guest-state area.

The value of the VMX-preemption timer is saved into the corresponding field in the guest-state area if the “save VMX-preemption timer value” VM-exit control is 1. That field becomes undefined if, in addition, either the SMM VM exit is from VMX root operation or the SMM VM exit is from VMX non-root operation and the “activate VMX-preemption timer” VM-execution control is 0.

34.15.2.5 Updating State

If an SMM VM exit is from VMX non-root operation and the “Intel PT uses guest physical addresses” VM-execution control is 1, the IA32_RTIT_CTL MSR is cleared to 00000000_00000000H.¹ This is done even if the “clear IA32_RTIT_CTL” VM-exit control is 0.

SMM VM exits affect the non-register state of a logical processor as follows:

- SMM VM exits cause non-maskable interrupts (NMIs) to be blocked; they may be unblocked through execution of IRET or through a VM entry (depending on the value loaded for the interruptibility state and the setting of the “virtual NMIs” VM-execution control).
- SMM VM exits cause SMIs to be blocked; they may be unblocked by a VM entry that returns from SMM (see Section 34.15.4).

SMM VM exits invalidate linear mappings and combined mappings associated with VPID 0000H for all PCIDs. Combined mappings for VPID 0000H are invalidated for all EPTRTA values (EPTRTA is the value of bits 51:12 of EPTP; see Section 31.4). (Ordinary VM exits are not required to perform such invalidation if the “enable VPID” VM-execution control is 1; see Section 30.5.5.)

34.15.3 Operation of the SMM-Transfer Monitor

Once invoked, the SMM-transfer monitor (STM) is in VMX root operation and can use VMX instructions to configure VMCSs and to cause VM entries to virtual machines supported by those structures. As noted in Section 34.15.1, the VMXOFF instruction cannot be used under the dual-monitor treatment and thus cannot be used by the STM.

The RSM instruction also cannot be used under the dual-monitor treatment. As noted in Section 28.1.3, it causes a VM exit if executed in SMM in VMX non-root operation. If executed in VMX root operation, it causes an invalid-opcode exception. The STM uses VM entries to return from SMM (see Section 34.15.4).

34.15.4 VM Entries that Return from SMM

The SMM-transfer monitor (STM) returns from SMM using a VM entry with the “entry to SMM” VM-entry control clear. VM entries that return from SMM reverse the effects of an SMM VM exit (see Section 34.15.2).

VM entries that return from SMM may differ from other VM entries in that they do not necessarily enter VMX non-root operation. If the executive-VMCS pointer field in the current VMCS contains the VMXON pointer, the logical processor remains in VMX root operation after VM entry.

For differences between VM entries that return from SMM and other VM entries see Sections 34.15.4.1 through 34.15.4.10.

34.15.4.1 Checks on the Executive-VMCS Pointer Field

VM entries that return from SMM perform the following checks on the executive-VMCS pointer field in the current VMCS:

- Bits 11:0 must be 0.
- The pointer must not set any bits beyond the processor’s physical-address width.^{2,3}

1. In this situation, the value of this MSR was saved earlier into the guest-state area. All VM exits save this MSR if the 1-setting of the “load IA32_RTIT_CTL” VM-entry control is supported (see Section 30.3.1), which must be the case if the “Intel PT uses guest physical addresses” VM-execution control is 1 (see Section 29.2.1.1).

2. Usually, the processor’s physical-address width is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), the width is reduced by the value of IA32_TME_ACTIVATE[39:36]. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

3. If IA32_VMX_BASIC[48] is read as 1, this pointer must not set any bits in the range 63:32; see Appendix A.1.

- The 32 bits located in memory referenced by the physical address in the pointer must contain the processor's VMCS revision identifier (see Section 27.2).

The checks above are performed before the checks described in Section 34.15.4.2 and before any of the following checks:

- If the "deactivate dual-monitor treatment" VM-entry control is 0 and the executive-VMCS pointer field does not contain the VMXON pointer, the launch state of the executive VMCS (the VMCS referenced by the executive-VMCS pointer field) must be launched (see Section 27.11.3).
- If the "deactivate dual-monitor treatment" VM-entry control is 1, the executive-VMCS pointer field must contain the VMXON pointer (see Section 34.15.7).¹

34.15.4.2 Checks on VM-Execution Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-execution control fields specified in Section 29.2.1.1. They do not apply the checks to the current VMCS. Instead, VM-entry behavior depends on whether the executive-VMCS pointer field contains the VMXON pointer:

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the checks are not performed at all.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the checks are performed on the VM-execution control fields in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS). These checks are performed after checking the executive-VMCS pointer field itself (for proper alignment).

Other VM entries ensure that, if "activate VMX-preemption timer" VM-execution control is 0, the "save VMX-preemption timer value" VM-exit control is also 0. This check is not performed by VM entries that return from SMM.

34.15.4.3 Checks on VM-Entry Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-entry control fields specified in Section 29.2.1.3.

Specifically, if the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the **injected-event identification** field must not indicate injection of a pending MTF VM exit (see Section 29.6.2). Specifically, the following cannot all be true for that field:

- the valid bit (bit 31) is 1
- the **event** type (bits 10:8) is 7 (other event); and
- the vector (bits 7:0) is 0 (pending MTF VM exit).

34.15.4.4 Checks on the Guest State Area

Section 29.3.1 specifies checks performed on fields in the guest-state area of the VMCS. Some of these checks are conditioned on the settings of certain VM-execution controls (e.g., "virtual NMIs" or "unrestricted guest").

VM entries that return from SMM modify these checks based on whether the executive-VMCS pointer field contains the VMXON pointer:²

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the checks are performed as all relevant VM-execution controls were 0. (As a result, some checks may not be performed at all.)
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), this check is performed based on the settings of the VM-execution controls in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS).

1. The STM can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

2. The STM can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

For VM entries that return from SMM, the activity-state field must not indicate the wait-for-SIPI state if the executive-VMCS pointer field contains the VMXON pointer (the VM entry is to VMX root operation).

34.15.4.5 Loading Guest State

VM entries that return from SMM load the SMBASE register from the SMBASE field.

VM entries that return from SMM invalidate linear mappings and combined mappings associated with all VPIDs. Combined mappings are invalidated for all EPTRTA values (EPTRTA is the value of bits 51:12 of EPTP; see Section 31.4). (Ordinary VM entries are required to perform such invalidation only for VPID 0000H and are not required to do even that if the “enable VPID” VM-execution control is 1; see Section 29.3.2.5.)

34.15.4.6 VMX-Preemption Timer

A VM entry that returns from SMM activates the VMX-preemption timer only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation) and the “activate VMX-preemption timer” VM-execution control is 1 in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field). In this case, VM entry starts the VMX-preemption timer with the value in the VMX-preemption timer-value field in the current VMCS.

34.15.4.7 Updating the Current-VMCS and SMM-Transfer VMCS Pointers

Successful VM entries (returning from SMM) load the SMM-transfer VMCS pointer with the current-VMCS pointer. Following this, they load the current-VMCS pointer from a field in the current VMCS:

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the current-VMCS pointer is loaded from the VMCS-link pointer field.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the current-VMCS pointer is loaded with the value of the executive-VMCS pointer field.

If the VM entry successfully enters VMX non-root operation, the VM-execution controls in effect after the VM entry are those from the new current VMCS. This includes any structures external to the VMCS referenced by VM-execution control fields.

The updating of these VMCS pointers occurs before event injection. Event injection is determined, however, by the VM-entry control fields in the VMCS that was current when the VM entry commenced.

34.15.4.8 VM Exits Induced by VM Entry

Section 29.6.1.2 describes how the event-delivery process invoked by event injection may lead to a VM exit. Section 29.7.3 to Section 29.7.7 describe other situations that may cause a VM exit to occur immediately after a VM entry.

Whether these VM exits occur is determined by the VM-execution control fields in the current VMCS. For VM entries that return from SMM, they can occur only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation).

In this case, determination is based on the VM-execution control fields in the VMCS that is current after the VM entry. This is the VMCS referenced by the value of the executive-VMCS pointer field at the time of the VM entry (see Section 34.15.4.7). This VMCS also controls the delivery of such VM exits. Thus, VM exits induced by a VM entry returning from SMM are to the executive monitor and not to the STM.

34.15.4.9 SMI Blocking

VM entries that return from SMM determine the blocking of system-management interrupts (SMIs) as follows:

- If the “deactivate dual-monitor treatment” VM-entry control is 0, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.
- If the “deactivate dual-monitor treatment” VM-entry control is 1, the blocking of SMIs depends on whether the logical processor is in SMX operation:¹
 - If the logical processor is in SMX operation, SMIs are blocked after VM entry.

- If the logical processor is outside SMX operation, SMIs are unblocked after VM entry.

VM entries that return from SMM and that do not deactivate the dual-monitor treatment may leave SMIs blocked. This feature exists to allow the STM to invoke functionality outside of SMM without unblocking SMIs.

34.15.4.10 Failures of VM Entries That Return from SMM

Section 29.8 describes the treatment of VM entries that fail during or after loading guest state. Such failures record information in the VM-exit information fields and load processor state as would be done on a VM exit. The VMCS used is the one that was current before the VM entry commenced. Control is thus transferred to the STM and the logical processor remains in SMM.

34.15.5 Enabling the Dual-Monitor Treatment

Code and data for the SMM-transfer monitor (STM) reside in a region of SMRAM called the **monitor segment** (MSEG). Code running in SMM determines the location of MSEG and establishes its content. This code is also responsible for enabling the dual-monitor treatment.

SMM code enables the dual-monitor treatment and specifies the location of MSEG by writing to the IA32_SMM_MONITOR_CTL MSR (index 9BH). The MSR has the following format:

- Bit 0 is the register's valid bit. The STM may be invoked using VMCALL only if this bit is 1. Because VMCALL is used to activate the dual-monitor treatment (see Section 34.15.6), the dual-monitor treatment cannot be activated if the bit is 0. This bit is cleared when the logical processor is reset.
- Bit 1 is reserved.
- Bit 2 determines whether executions of VMXOFF unblock SMIs under the default treatment of SMIs and SMM. Executions of VMXOFF unblock SMIs unless bit 2 is 1 (the value of bit 0 is irrelevant). See Section 34.14.4. Certain leaf functions of the GETSEC instruction clear this bit (see Chapter 7, "Safer Mode Extensions Reference," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D).
- Bits 11:3 are reserved.
- Bits 31:12 contain a value that, when shifted left 12 bits, is the physical address of MSEG (the MSEG base address).
- Bits 63:32 are reserved.

The following items detail use of this MSR:

- The IA32_SMM_MONITOR_CTL MSR is supported only on processors that support the dual-monitor treatment.¹ On other processors, accesses to the MSR using RDMSR or WRMSR generate a general-protection fault (#GP(0)).
- A write to the IA32_SMM_MONITOR_CTL MSR using WRMSR generates a general-protection fault (#GP(0)) if executed outside of SMM or if an attempt is made to set any reserved bit. An attempt to write to the IA32_SMM_MONITOR_CTL MSR fails if made as part of a VM exit that does not end in SMM or part of a VM entry that does not begin in SMM.
- Reads from the IA32_SMM_MONITOR_CTL MSR using RDMSR are allowed any time RDMSR is allowed. The MSR may be read as part of any VM exit.
- The dual-monitor treatment can be activated only if the valid bit in the MSR is set to 1.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 7, "Safer Mode Extensions Reference," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D.

1. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether the dual-monitor treatment is supported.

The 32 bytes located at the MSEG base address are called the **MSEG header**. The format of the MSEG header is given in Table 34-10 (each field is 32 bits).

Table 34-10. Format of MSEG Header

Byte Offset	Field
0	MSEG-header revision identifier
4	SMM-transfer monitor features
8	GDTR limit
12	GDTR base offset
16	CS selector
20	EIP offset
24	ESP offset
28	CR3 offset

To ensure proper behavior in VMX operation, software should maintain the MSEG header in writeback cacheable memory. Future implementations may allow or require a different memory type.¹ Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

SMM code should enable the dual-monitor treatment (by setting the valid bit in IA32_SMM_MONITOR_CTL MSR) only after establishing the content of the MSEG header as follows:

- Bytes 3:0 contain the **MSEG revision identifier**. Different processors may use different MSEG revision identifiers. These identifiers enable software to avoid using an MSEG header formatted for one processor on a processor that uses a different format. Software can discover the MSEG revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).
- Bytes 7:4 contain the **SMM-transfer monitor features** field. Bits 31:1 of this field are reserved and must be zero. Bit 0 of the field is the **IA-32e mode SMM feature bit**. It indicates whether the logical processor will be in IA-32e mode after the STM is activated (see Section 34.15.6).
- Bytes 31:8 contain fields that determine how processor state is loaded when the STM is activated (see Section 34.15.6.5). SMM code should establish these fields so that activating of the STM invokes the STM's initialization code.

34.15.6 Activating the Dual-Monitor Treatment

The dual-monitor treatment may be enabled by SMM code as described in Section 34.15.5. The dual-monitor treatment is activated only if it is enabled and only by the executive monitor. The executive monitor activates the dual-monitor treatment by executing VMCALL in VMX root operation.

When VMCALL activates the dual-monitor treatment, it causes an SMM VM exit. Differences between this SMM VM exit and other SMM VM exits are discussed in Sections 34.15.6.1 through 34.15.6.6. See also “VMCALL—Call to VM Monitor” in Chapter 33.

34.15.6.1 Initial Checks

An execution of VMCALL attempts to activate the dual-monitor treatment if (1) the processor supports the dual-monitor treatment;² (2) the logical processor is in VMX root operation; (3) the logical processor is outside SMM and

1. Alternatively, software may map the MSEG header with the UC memory type; this may be necessary, depending on how memory is organized. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with exceptions noted in Appendix A.1.

the valid bit is set in the IA32_SMM_MONITOR_CTL MSR; (4) the logical processor is not in virtual-8086 mode and not in compatibility mode; (5) CPL = 0; and (6) the dual-monitor treatment is not active.

Such an execution of VMCALL begins with some initial checks. These checks are performed before updating the current-VMCS pointer and the executive-VMCS pointer field (see Section 34.15.2.2).

The VMCS that manages SMM VM exit caused by this VMCALL is the current VMCS established by the executive monitor. The VMCALL performs the following checks on the current VMCS in the order indicated:

1. There must be a current VMCS pointer.
2. The launch state of the current VMCS must be clear.
3. The VM-exit controls in the current VMCS must be set properly:
 - Reserved bits in the primary VM-exit controls must be set properly. Software may consult the VMX capability MSRs to determine the proper setting (see Appendix A.4.1).
 - If the “activate secondary controls” primary VM-exit control is 1, reserved bits in the secondary VM-exit controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.4.2).
 - If the “activate secondary controls” primary VM-exit control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary VM-exit controls. The logical processor operates as if all the secondary VM-exit controls were 0.

If any of these checks fail, subsequent checks are skipped and VMCALL fails. If all these checks succeed, the logical processor uses the IA32_SMM_MONITOR_CTL MSR to determine the base address of MSEG. The following checks are performed in the order indicated:

1. The logical processor reads the 32 bits at the base of MSEG and compares them to the processor’s MSEG revision identifier.
2. The logical processor reads the SMM-transfer monitor features field:
 - Bit 0 of the field is the IA-32e mode SMM feature bit, and it indicates whether the logical processor will be in IA-32e mode after the SMM-transfer monitor (STM) is activated.
 - If the VMCALL is executed on a processor that does not support Intel 64 architecture, the IA-32e mode SMM feature bit must be 0.
 - If the VMCALL is executed in 64-bit mode, the IA-32e mode SMM feature bit must be 1.
 - Bits 31:1 of this field are currently reserved and must be zero.

If any of these checks fail, subsequent checks are skipped and the VMCALL fails.

34.15.6.2 Updating the Current-VMCS and Executive-VMCS Pointers

Before performing the steps in Section 34.15.2.2, SMM VM exits that activate the dual-monitor treatment begin by loading the SMM-transfer VMCS pointer with the value of the current-VMCS pointer.

34.15.6.3 Saving Guest State

As noted in Section 34.15.2.4, SMM VM exits save the contents of the SMBASE register into the corresponding field in the guest-state area. While this is true also for SMM VM exits that activate the dual-monitor treatment, the VMCS used for those VM exits exists outside SMRAM.

The SMM-transfer monitor (STM) can also discover the current value of the SMBASE register by using the RDMSR instruction to read the IA32_SMBASE MSR (MSR address 9EH). The following items detail use of this MSR:

- The MSR is supported only if IA32_VMX_MISC[15] = 1 (see Appendix A.6).
- A write to the IA32_SMBASE MSR using WRMSR generates a general-protection fault (#GP(0)). An attempt to write to the IA32_SMBASE MSR fails if made as part of a VM exit or part of a VM entry.

-
2. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether the dual-monitor treatment is supported.

- A read from the IA32_SMBASE MSR using RDMSR generates a general-protection fault (#GP(0)) if executed outside of SMM. An attempt to read from the IA32_SMBASE MSR fails if made as part of a VM exit that does not end in SMM.

34.15.6.4 Saving MSRs

The VM-exit MSR-store area is not used by SMM VM exits that activate the dual-monitor treatment. No MSRs are saved into that area.

34.15.6.5 Loading Host State

The VMCS that is current during an SMM VM exit that activates the dual-monitor treatment was established by the executive monitor. It does not contain the VM-exit controls and host state required to initialize the STM. For this reason, such SMM VM exits do not load processor state as described in Section 30.5. Instead, state is set to fixed values or loaded based on the content of the MSEG header (see Table 34-10):

- CR0 is set to as follows:
 - PG, NE, ET, MP, and PE are all set to 1.
 - CD and NW are left unchanged.
 - All other bits are cleared to 0.
- CR3 is set as follows:
 - Bits 63:32 are cleared on processors that support IA-32e mode.
 - Bits 31:12 are set to bits 31:12 of the sum of the MSEG base address and the CR3-offset field in the MSEG header.
 - Bits 11:5 and bits 2:0 are cleared (the corresponding bits in the CR3-offset field in the MSEG header are ignored).
 - Bits 4:3 are set to bits 4:3 of the CR3-offset field in the MSEG header.
- CR4 is set as follows:
 - MCE, PGE, CET, PCIDE, LA57, and FRED are cleared.
 - PAE is set to the value of the IA-32e mode SMM feature bit.
 - If the IA-32e mode SMM feature bit is clear, PSE is set to 1 if supported by the processor; if the bit is set, PSE is cleared.
 - All other bits are unchanged.
- DR7 is set to 400H.
- The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.
- The registers CS, SS, DS, ES, FS, and GS are loaded as follows:
 - All registers are usable.
 - CS.selector is loaded from the corresponding field in the MSEG header (the high 16 bits are ignored), with bits 2:0 cleared to 0. If the result is 0000H, CS.selector is set to 0008H.
 - The selectors for SS, DS, ES, FS, and GS are set to CS.selector+0008H. If the result is 0000H (if the CS selector was FFF8H), these selectors are instead set to 0008H.
 - The base addresses of all registers are cleared to zero.
 - The segment limits for all registers are set to FFFFFFFFH.
 - The AR bytes for the registers are set as follows:
 - CS.Type is set to 11 (execute/read, accessed, non-conforming code segment).
 - For SS, DS, ES, FS, and GS, the Type is set to 3 (read/write, accessed, expand-up data segment).
 - The S bits for all registers are set to 1.
 - The DPL for each register is set to 0.

- The P bits for all registers are set to 1.
- On processors that support Intel 64 architecture, CS.L is loaded with the value of the IA-32e mode SMM feature bit.
- CS.D is loaded with the inverse of the value of the IA-32e mode SMM feature bit.
- For each of SS, DS, ES, FS, and GS, the D/B bit is set to 1.
- The G bits for all registers are set to 1.
- LDTR is unusable. The LDTR selector is cleared to 0000H, and the register is otherwise undefined (although the base address is always canonical)
- GDTR.base is set to the sum of the MSEG base address and the GDTR base-offset field in the MSEG header (bits 63:32 are always cleared on processors that support IA-32e mode). GDTR.limit is set to the corresponding field in the MSEG header (the high 16 bits are ignored).
- IDTR.base is unchanged. IDTR.limit is cleared to 0000H.
- RIP is set to the sum of the MSEG base address and the value of the RIP-offset field in the MSEG header (bits 63:32 are always cleared on logical processors that support IA-32e mode).
- RSP is set to the sum of the MSEG base address and the value of the RSP-offset field in the MSEG header (bits 63:32 are always cleared on logical processor that supports IA-32e mode).
- RFLAGS is cleared, except bit 1, which is always set.
- The logical processor is left in the active state.
- Event blocking after the SMM VM exit is as follows:
 - There is no blocking by STI or by MOV SS.
 - There is blocking by non-maskable interrupts (NMIs) and by SMIs.
- There are no pending debug exceptions after the SMM VM exit.
- For processors that support IA-32e mode, the IA32_EFER MSR is modified so that LME and LMA both contain the value of the IA-32e mode SMM feature bit.

If any of CR3[63:5], CR4.PAE, CR4.PSE, or IA32_EFER.LMA is changing, the TLBs are updated so that, after VM exit, the logical processor does not use translations that were cached before the transition. This is not necessary for changes that would not affect paging due to the settings of other bits (for example, changes to CR4.PSE if IA32_EFER.LMA was 1 before and after the transition).

34.15.6.6 Loading MSRs

The VM-exit MSR-load area is not used by SMM VM exits that activate the dual-monitor treatment. No MSRs are loaded from that area.

34.15.7 Deactivating the Dual-Monitor Treatment

The SMM-transfer monitor may deactivate the dual-monitor treatment and return the processor to default treatment of SMIs and SMM (see Section 34.14). It does this by executing a VM entry with the “deactivate dual-monitor treatment” VM-entry control set to 1.

As noted in Section 29.2.1.3 and Section 34.15.4.1, an attempt to deactivate the dual-monitor treatment fails in the following situations: (1) the processor is not in SMM; (2) the “entry to SMM” VM-entry control is 1; or (3) the executive-VMCS pointer does not contain the VMXON pointer (the VM entry is to VMX non-root operation).

As noted in Section 34.15.4.9, VM entries that deactivate the dual-monitor treatment ignore the SMI bit in the interruptibility-state field of the guest-state area. Instead, the blocking of SMIs following such a VM entry depends on whether the logical processor is in SMX operation:¹

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 7, “Safer Mode Extensions Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

- If the logical processor is in SMX operation, SMIs are blocked after VM entry. SMIs may later be unblocked by the VMXOFF instruction (see Section 34.14.4) or by certain leaf functions of the GETSEC instruction (see Chapter 7, “Safer Mode Extensions Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D).
- If the logical processor is outside SMX operation, SMIs are unblocked after VM entry.

34.16 SMI AND PROCESSOR EXTENDED STATE MANAGEMENT

On processors that support processor extended states using XSAVE/XRSTOR (see Chapter 13, “Managing State Using the XSAVE Feature Set,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1), the processor does not save any XSAVE/XRSTOR related state on an SMI. It is the responsibility of the SMI handler code to properly preserve the state information (including CR4.OSXSAVE, XCR0, and possibly processor extended states using XSAVE/XRSTOR). Therefore, the SMI handler must follow the rules described in Chapter 13, “Managing State Using the XSAVE Feature Set,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

34.17 MODEL-SPECIFIC SYSTEM MANAGEMENT ENHANCEMENT

This section describes enhancement of system management features that apply only to the 4th generation Intel Core processors. These features are model-specific. BIOS and SMM handler must use CPUID to enumerate Display-Family_DisplayModel signature when programming with these interfaces.

34.17.1 SMM Handler Code Access Control

The BIOS may choose to restrict the address ranges of code that SMM handler executes. When SMM handler code execution check is enabled, an attempt by the SMM handler to execute outside the ranges specified by SMRR (see Section 34.4.2.1) will cause the assertion of an unrecoverable machine check exception (MCE).

The interface to enable SMM handler code access check resides in a per-package scope model-specific register MSR_SMM_FEATURE_CONTROL at address 4E0H. An attempt to access MSR_SMM_FEATURE_CONTROL outside of SMM will cause a #GP. Writes to MSR_SMM_FEATURE_CONTROL is further protected by configuration interface of MSR_SMM_MCA_CAP at address 17DH.

Details of the interface of MSR_SMM_FEATURE_CONTROL and MSR_SMM_MCA_CAP are described in Table 2-29 in Chapter 2, “Model-Specific Registers (MSRs),” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

34.17.2 SMI Delivery Delay Reporting

Entry into the system management mode occurs at instruction boundary. In situations where a logical processor is executing an instruction involving a long flow of internal operations, servicing an SMI by that logical processor will be delayed. Delayed servicing of SMI of each logical processor due to executing long flows of internal operation in a physical processor can be queried via a package-scope register MSR_SMM_DELAYED at address 4E2H.

The interface to enable reporting of SMI delivery delay due to long internal flows resides in a per-package scope model-specific register MSR_SMM_DELAYED. An attempt to access MSR_SMM_DELAYED outside of SMM will cause a #GP. Availability to MSR_SMM_DELAYED is protected by configuration interface of MSR_SMM_MCA_CAP at address 17DH.

Details of the interface of MSR_SMM_DELAYED and MSR_SMM_MCA_CAP are described in Table 2-29 in Chapter 2, “Model-Specific Registers (MSRs),” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

34.17.3 Blocked SMI Reporting

A logical processor may have entered into a state and blocked from servicing other interrupts (including SMI). Logical processors in a physical processor that are blocked in serving SMI can be queried in a package-scope register MSR_SMM_BLOCKED at address 4E3H. An attempt to access MSR_SMM_BLOCKED outside of SMM will cause a #GP.

Details of the interface of MSR_SMM_BLOCKED is described in Table 2-29 in Chapter 2, “Model-Specific Registers (MSRs),” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

27. Updates to Chapter 35, Volume 3C

Change bars and text show changes to Chapter 35 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- In Section 35.3.1, noted that SEAMRET instructions are described in the Intel® Trust Domain CPU Architectural Extensions document.
- In the instruction SEAMCALL—Enter SEAM Root Operation, corrected the 64-bit Mode to valid and Compatibility/Legacy Mode to invalid.
- In the description of SEAMCALL—Enter SEAM Root Operation, clarified that SEAMCALL can be executed by VMX in root or non-root operation.
- In the SEAMCALL instruction, corrected reference for VMFailInvalid indication to Section 33.2 and reference for SEAMCALL Flags Affected to Section 33.2.
- In the instruction TDCALL—Call SEAM Module, corrected the 64-bit Mode to valid.
- In the Flags Affected section of instruction TDCALL—Call SEAM Module, removed reference to Section 32.2 since TDCALL never returns VMFailInvalid.
- Updated conditions for Virtual-8086 Mode Exceptions for TDCALL.

CHAPTER 35 SECURE ARBITRATION MODE (SEAM)

This chapter specifies secure arbitration mode (SEAM), an extension of the virtual machines extensions architecture (VMX) that allows running secure virtual machines.

The SEAM architecture is used as a foundation for Trusted Domains Extensions (TDX) architecture. For details about TDX, see *Intel® Trust Domain Extensions (Intel® TDX) — An overview of the Intel TDX technology*.

35.1 MODES OF OPERATION

The SEAM architecture defines a protected mode of operation called SEAM, which is an extension of VMX operation. Like VMX operation, SEAM includes root and non-root operations:

- **SEAM root operation** is the SEAM extension of VMX root operation. The software running in SEAM root operation is called a **SEAM module**. There are two sub-modes to SEAM root operation:
 - **Loader mode:** At most one logical processor can execute in this sub-mode at any time.
 - **Module mode:** Multiple logical processors can execute in this sub-mode in parallel.
- **SEAM non-root operation** is the SEAM extension of VMX non-root operation. The software running in this SEAM non-root operation is called **SEAM guest**. Multiple logical processors can execute in this mode concurrently.

Certain platform operations may be required to enable SEAM; until they are performed, SEAM may be globally disabled. Once SEAM is enabled, it may or may not be ready for execution in loader mode or in module mode (again, possibly based on certain platform operations). Details regarding this enabling are provided in the *Intel® Trust Domain CPU Architectural Extensions*.

The following diagram shows the VMX and SEAM software components, the modes in which they operate, and the transitions between these execution modes.

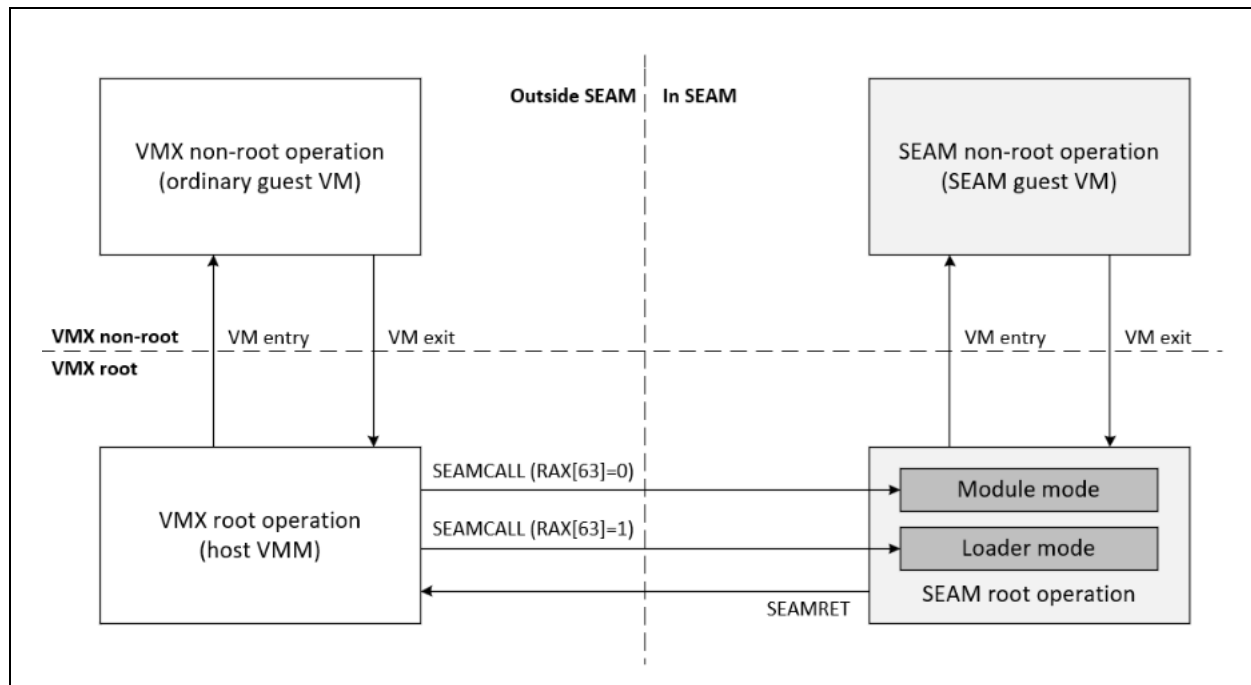


Figure 35-1. SEAM Modes of Operation

35.1.1 SEAM Calls

Processor transitions from VMX root operation to SEAM root operation are called **SEAM calls**. SEAM calls are initiated by executing the SEAMCALL instruction. A SEAM call operates in a manner similar to that of SMM VM exits (see Section 34.15.2) in response to execution of the VMCALL instruction in VMX root operation. A SEAM call uses a **SEAM-transfer VMCS** that resides within the SEAM range of the physical-address space (see Section 35.4.1).

SEAM calls use RAX[63] to determine the SEAM root operation sub-mode to be entered. When RAX[63] is 0, the transition is into module-mode SEAM root operation; otherwise, it is into loader-mode SEAM root operation. An attempt to enter loader-mode SEAM root operation fails if another logical processor is in SEAM having entered in loader mode.

35.1.2 SEAM Returns

Processor transitions from SEAM root operation back to VMX root operation are called **SEAM returns**. SEAM returns are initiated by executing the SEAMRET instruction. A SEAM return operates in a manner similar to that of VM entries that return from SMM (see Section 34.15.4). Like SEAM calls, SEAM returns use a SEAM-transfer VMCS.

A SEAM return from loader-mode SEAM root operation ensures that all data for any VMCS that the logical processor used in SEAM (including the SEAM-transfer VMCS) are in memory; in addition, there is no current VMCS after such a SEAM return.

35.1.3 SEAM Host-Guest Transitions

Processor transitions between SEAM root operation and SEAM non-root operation are similar to processor transition between VMX root operation and VMX non-root operation. A VMCS used to transition between SEAM root and non-root operation, and to control SEAM non-root operation, is called **SEAM-guest VMCS**.

A SEAM-guest VMCS supports SEAM-specific fields and controls. In general, those fields and controls are ignored outside SEAM.

The SEAM-specific VMCS fields are the following:

- **SEAM-guest KeyID**. This is a VM-execution control field that contains the key identifier (KeyID) assigned for the key with which the SEAM guest's private pages are encrypted in memory.
- **SEAM shared EPT pointer**. This is a VM-execution control field that contains the physical address of the base of shared EPT PML4 table or PML5 table (depending on how EPT is configured).

The SEAM-specific VMX controls are the following:

- **SEAM guest-physical address width**. This is a VM-execution control that determines, for SEAM non-root operation, the width of guest-physical addresses translated by EPT and also the position of the SHARED bit in guest-physical addresses. See Section 35.4.
- **Allow SEAM-guest telemetry**. This is a VM-entry control that allows collection of core telemetry in SEAM non-root operation.

Any processor that supports SEAM also supports the SEAM-specific VMCS fields and controls identified above. Support for the SEAM-specific VMX controls above is enumerated normally by the IA32_VMX_PROCBASED_CTLSS3 MSR (for "SEAM guest-physical address width") and by IA32_VMX_ENTRY_CTLSS MSR (for "allow SEAM-guest telemetry").

SEAM guests must run with EPT enabled. VM entry in SEAM root operation will fail if the "enable EPT" VM-execution control is not set in the SEAM-guest VMCS.

35.2 SEAM AND TME-MK

SEAM interoperates with multi-key total memory encryption (TME-MK). This section explains the specifics of TME-MK related to that interoperation.

The current configuration of TME-MK can be inferred from the contents of the IA32_TME_ACTIVATE MSR (MSR address 982H). The following fields are relevant:

- If `IA32_TME_ACTIVATE[1:0] = 11B`, total-memory encryption (TME) is active.
- `IA32_TME_ACTIVATE.MK_TME_KEYID_BITS[35:32]` specifies the number of bits in a physical address that are interpreted as a key identifier (KeyID). If TME is active and this value is greater than 0, TME-MK is active.
- `IA32_TME_ACTIVATE.TDX_RESERVED_KEYID_BITS[39:36]` specifies the number of KeyID bits that can be used only in SEAM-private KeyIDs (see Section 35.3.2). This value never exceeds `IA32_TME_ACTIVATE.MK_TME_KEYID_BITS`.

Let m be the value enumerated by `CPUID.80000008H:EAX[7:0]`; $k = \text{IA32\_TME\_ACTIVATE.MK\_TME\_KEYID\_BITS}$; and $p = \text{IA32\_TME\_ACTIVATE.TDX\_RESERVED\_KEYID\_BITS}$. Then in any physical address, bits $m-1:m-k$ form a key identifier that is used to select an encryption key (only bits $(m-k)-1:0$ are used to address memory). The `PCONFIG` instruction can be used to specify the encryption key associated with each key identifier. It sets it sets

The `IA32_MKTME_KEYID_PARTITIONING` MSR (MSR address 87H) enumerates the number of key identifiers available for use:

- `IA32_MKTME_KEYID_PARTITIONING.NUM_MKTME_KIDS[31:0]` enumerates the number of TME-MK key identifiers available for general use. These are numbered 1 to `NUM_MKTME_KIDS` (0 is not considered a TME-MK key identifier).
- `IA32_MKTME_KEYID_PARTITIONING.NUM_TDX_PRIV_KIDS[63:32]` enumerates the number of SEAM-private key identifiers. These are numbered `NUM_MKTME_KIDS + 1` to `NUM_MKTME_KIDS + NUM_TDX_PRIV_KIDS`.

35.3 OPERATION IN SEAM

This section provides details of both SEAM root operation and SEAM non-root operation.

35.3.1 SEAM Root Operation

Processor behavior in SEAM root operation is similar to processor behavior in VMX root operation, with the following differences:

- Software can execute `SEAMRET` and `SEAMOPS`. (Execution of these instructions outside SEAM root operation results in a `#UD` exception.) As noted in Section 35.1.2, execution of `SEAMRET` causes a SEAM return to VMX root operation. The [SEAMRET and SEAMOPS instructions](#) are described in *Intel® Trust Domain CPU Architectural Extensions*.
- Software can access a special set of SEAM-only MSRs. An attempt to access a SEAM-only MSR outside SEAM results in a `#GP(0)` exception. (Some may be accessible only in SEAM root operation.) The full set of SEAM-only MSRs is specified in *Intel® Trust Domain CPU Architectural Extensions*.
- An attempt to execute the `VMXOFF` instruction or the `VMCALL` instruction in SEAM root operation results in a `#GP(0)` exception. Software in SEAM cannot leave VMX operation, nor can it invoke an SMM-transfer monitor using an SMM VM exit.
- Entry to SEAM root operation (by either a SEAM call or a VM exit from SEAM non-root operation) blocks NMIs. Execution of `IRET` in SEAM root operation unblocks NMIs normally.
- System-management interrupts (SMIs), uncore machine checks and machine-check SMIs (MSMIs) are masked in SEAM root operation. If pending, they are delivered after leaving SEAM root operation.
- Triple faults and machine checks caused by poisoned cache-line consumption in SEAM root operation cause unbreakable shutdown.

35.3.2 SEAM Non-Root Operation

Processor behavior in SEAM non-root operation is largely the same as that in VMX non-root operation.

System-management interrupts (SMIs) and machine-check SMIs (MSMIs) that occur in SEAM non-root operation cause a VM exit instead of being delivered normally. The SMI or MSMI remains pending and will be delivered following the next SEAM return.

The remainder of this section explains the changes in SEAM non-root operation with regard to the use of guest-physical addresses.

As noted in Section 35.1.3, VM entry to SEAM non-root operation requires EPT to be enabled. Thus, all addresses used by guest software as physical addresses (e.g., addresses in the guest's paging structures) are treated as guest-physical addresses and are thus translated by EPT. Translation by EPT in SEAM non-root operation differs somewhat from that in ordinary VMX non-root operation.

In VMX non-root operation, the width of guest-physical addresses is either 48 bits (if 4-level EPT is in use) or 52 bits (for 5-level EPT). In SEAM non-root operation, the guest-physical address width is 48 bits if the "SEAM guest-physical address width" VM-execution control is 0 (even if 5-level EPT is in use).

If the guest-physical address width is 48, an access to a guest-physical address that sets bits in the range 51:48 causes an EPT violation. (This is not the case with 5-level EPT outside SEAM. In SEAM, this occurs with 5-level paging as long as the "SEAM guest-physical address width" VM-execution control is 0.)

The uppermost bit in a guest-physical address (bit 47 or bit 51 as appropriate) is defined to be the address's "SHARED" bit. Guest-physical addresses that clear this bit are **SEAM-guest private addresses**; those that set this bit are **SEAM-guest shared addresses**.

Certain situations in SEAM non-root operation require SEAM-guest private addresses:

- Instruction fetches are limited to SEAM-guest private addresses. An attempt to fetch an instruction from a SEAM-guest shared address results in a general-protection exception (#GP(0)).
- CR3 must contain a SEAM-guest private address. An attempt to load CR3 with a SEAM-guest shared address results in #GP(0).
- With PAE paging, each of the PDPTs registers must contain a SEAM-guest private address. An attempt to load any PDPT register with a SEAM-guest shared address results in #GP(0).
- The guest paging structures must be located at SEAM-guest private addresses. If linear-address translation encounters a paging-structure entry that references a paging structure with a SEAM-guest shared address, a page fault (#PF) occurs. The error code will set bit 3, indicating that a reserved bit was set.

Translation of guest-physical addresses differs for private and shared addresses:

- Translation of a SEAM-guest private address uses EPT paging structures located using the EPTP field in the VMCS.
 - Any physical address used to access an EPT paging structure is modified as follows. Let k be the value of `IA32_TME_ACTIVATE.MK_TME_KEYID_BITS[35:32]` and M be the value enumerated by `CPUID.80000008H:EAX[7:0]`. Before being used to access memory, bits $M-1:M-k$ of the physical address are replaced by bits $k-1:0$ of the SEAM-guest KeyID field in the VMCS.
 - The final physical address (to which the original SEAM-guest private address is being translated) is also modified in the manner described in the previous paragraph.
- Translation of a SEAM-guest shared address uses EPT paging structures located using the SEAM shared EPT pointer in the VMCS.
 - Bits 7:0 of the EPTP field (not the SEAM shared EPT pointer) control the translation of the guest-physical address.
 - Any physical address used to access an EPT paging structure should not use a SEAM-private KeyID (see Section 35.2).

If translation of a SEAM-guest shared address would access an EPT paging structure with a SEAM-private KeyID, an EPT misconfiguration occurs.

 - The final physical address (to which the original SEAM-guest shared address is being translated) also must not contain a SEAM-private KeyID; otherwise, an EPT misconfiguration occurs.

35.4 MEMORY PROTECTION

This section specifies access controls and cryptographic measures applied to protect software running in SEAM root operation (SEAM modules) and SEAM non-root operation (SEAM guests).

35.4.1 SEAM Range

The SEAM range is a physical memory range sequestered for SEAM modules. The range is allocated using range-register MSRs IA32_SEAMRR_BASE (index 1400H) and IA32_SEAMRR_MASK (index 1401H). The existence of these MSRs is enumerated by bit 15 of IA32_MTRRCAP (MSR address FEH). Processors that support SEAM architecture return IA32_MTRRCAP bit 15 as 1. For more information about these MSRs, see Chapter 2 in Volume 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*.

The SEAM range is divided into a **module sub-range** and a **loader sub-range**.

Bit 11 of IA32_SEAMRR_MASK MSR indicates whether the SEAM range is valid and protected. When this bit is 1, the processor provides the SEAM range with cryptographic and access-control protections as follows:

- Outside SEAM root operation, accesses to the physical addresses in the SEAM range are aborted, meaning that writes are ignored and reads return values with all bits set to 1.
- The following items apply in SEAM root operation:
 - Accesses to the module sub-range of the SEAM range are allowed.
 - Accesses to the loader sub-range of the SEAM range are allowed only in loader-mode SEAM root operation; otherwise they are aborted (see above).
 - Instruction fetches from physical addresses outside the SEAM range are prevented by the processor (the response to such attempts is implementation-specific).
- The contents of the SEAM range are encrypted by the TME-MK engine. The encryption key is machine-specific. It's typically an "ephemeral" key that is generated each time the platform is restarted.

Accesses to SEAM range when CR0.CD = 0 use the writeback (WB) memory type regardless of the MTRR and PAT registers. If CR0.CD = 1, accesses use the uncacheable (UC) memory type.

35.4.2 Memory Protections

Outside SEAM, registers holding physical addresses cannot be loaded with physical addresses with SEAM-private KeyIDs (see Section 35.3.2):

- Attempts to load control registers and MSRs with a physical address with a SEAM-private KeyID results in a general-protection exception. This does not apply to loads of the IA32_RTIT_OUTPUT_BASE MSR (using WRMSR, WRMSRLIST, WRMSRNS, or XRSTORS), as that MSR is checked at another time (see below).
- Linear-address translation without EPT that encounters a paging-structure entry with a physical address with a SEAM-private KeyID results in a page fault.
- Guest-physical-address translation with EPT that encounters an EPT paging-structure entry with a physical address with a SEAM-private KeyID results in an EPT misconfiguration.
- VM entry fails if it would use or load a physical address (from a VMCS field other than the SEAM shared EPT pointer) that contains a SEAM-private KeyID.
- The value of the physical address loaded into IA32_RTIT_OUTPUT_BASE MSR is checked when processor trace is enabled (when IA32_RTIT_CTL.TraceEn is set to 1), as is the physical address of a new output buffer when it is loaded from the table of physical addresses (ToPA).

If, despite the limitations identified above, there is an access using a physical address with a SEAM-private KeyID (perhaps because an MSR had been loaded before TME was activated), the access is aborted, meaning that writes are ignored and reads return values with all bits set to 1.

A cacheline that was written with a physical address with a SEAM-private KeyID cannot be read using a physical address with a different KeyID. The memory-protection mode supported by a particular processor is machine-dependent. For information about memory protection modes, see *Intel® Architecture Memory Protections for Confidential Compute Usages*.

35.5 SEAM INSTRUCTION REFERENCE

The SEAM architecture includes the following instructions:

- **SEAMCALL.** Execution causes a SEAM call (see Section 35.1.1) to enter SEAM root operation. It can be used by virtual-machine monitors (operating in VMX root operation) to call SEAM module services.
- **TDCALL.** Execution causes a VM exit from SEAM non-root operation to SEAM root operation. It can be used by SEAM guests to request SEAM-module services.
- **SEAMOPS.** Execution invokes SEAM-specific operation. It is used by SEAM modules to execute such operations.
- **SEAMRET.** Execution causes a SEAM return (Section 35.1.2) that leaves SEAM root operation. It is used by SEAM modules to return to the calling virtual-machine monitor.

SEAMCALL and TDCALL are described later in this section. SEAMOPS and SEAMRET can be executed only in SEAM root operation. Details are given in *Intel® Trust Domain CPU Architectural Extensions*, while the following items provide high-level information:

- **SEAMOPS.** The SEAMOPS instruction allows software running in SEAM root operation to invoke SEAM-specific operations. Some of these operations can be invoked only in loader mode.
- **SEAMRET.**
 - SEAMRET is a privileged instruction that allows software running in SEAM root operation to return to VMX root operation.
 - The instruction causes a SEAM return (see Section 35.1.2) using the SEAM-transfer VMCS. This transition is similar to VM entries that return from SMM (see Section 34.15.4).

An attempt to execute a SEAM instruction on a processor that does not support SEAM results in an invalid-opcode exception (#UD).

SEAMCALL—Enter SEAM Root Operation

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
66 0F 01 CF	SEAMCALL	Z0	Valid	Invalid	Transitions to SEAM root operation.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

SEAMCALL is a privileged instruction that passes control to software running in SEAM root operation. The instruction can be executed only in 64-bit mode in VMX operation outside SEAM and outside SMM. *In VMX non-root operation, execution causes a VM exit with exit reason 76, SEAMCALL.*

Execution when the SEAM range is not valid (IA32_SEAMRR_MASK MSR[11] = 0) or immediately following execution of MOV to SS or POP SS instruction results in a general-protection exception (#GP).

The instruction uses RAX[63] to select a sub-mode of SEAM root operation: if this value is 1, the instruction transitions into loader-mode SEAM root operation; otherwise, it transitions into module-mode SEAM root operation.

The instruction fails with VMFailInvalid indication (see [Section 33.2](#)) if SEAM is globally disabled; or if a loader-mode SEAM root operation is called when another logical processor is in SEAM having entered in loader mode.

If there is no failure, the SEAMCALL uses the SEAM-transfer VMCS to effect a SEAM call (see [Section 35.1.1](#)). The details of such transitions are specified in the *Intel® Trust Domain CPU Architectural Extensions*.

Operation

```

IF not in VMX operation or in SMM or in SEAM or IA32_EFER.LMA = 0 or CS.L = 0
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF IA32_SEAMRR_MASK.VALID = 0 or events blocked by MOV SS
    THEN #GP(0);
ELSIF SEAM is globally disabled
    THEN VMFailInvalid;
ELSIF RAX[63] = 0
    THEN
        IF module-mode SEAM root operation is not ready for execution
            THEN VMFailInvalid;
        ELSE
            Enter module-mode SEAM root operation;
        FI;
    ELSE
        IF loader-mode SEAM root operation is not ready for execution or
            another logical processor is executing in SEAM having entered in loader mode
            THEN VMFailInvalid;
        ELSE
            Enter loader-mode SEAM root operation;
        FI;
    FI;

```


Flags Affected

See the Operation section above and [Section 33.2](#).

Protected Mode Exceptions

#UD SEAMCALL is not recognized in protected mode.

Real-Address Mode Exceptions

#UD SEAMCALL is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD SEAMCALL is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD SEAMCALL is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the SEAM range is not valid. If events blocked by MOV SS.
#UD	If executed outside VMX operation. If executed in system-management mode (SMM). If executed in SEAM.

TDCALL—Call SEAM Module

Op/Code	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
66 0F 01 CC	TDCALL	Z0	Valid	Valid	Call SEAM monitor from SEAM guest.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

TDCALL is a privileged instruction that allows software running in SEAM non-root operation to call software running in SEAM root operation. The instruction can be executed only in SEAM non-root operation or VMX non-root operation.

In SEAM non-root operation or VMX non-root operation, execution causes a VM exit with exit reason 77 (TDCALL).

Operation

```
IF not in SEAM non-root operation and not in VMX non-root operation
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
ELSE VM exit;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#UD If executed outside VMX non-root operation and outside SEAM non-root operation.
 #GP(0) If the current privilege level is not 0.

Real-Address Mode Exceptions

#UD If executed outside VMX non-root operation and outside SEAM non-root operation.

Virtual-8086 Mode Exceptions

#UD If executed outside VMX non-root operation and outside SEAM non-root operation.
 #GP(0) If executed in VMX non-root operation or in SEAM non-root operation.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

Same as protected mode exceptions.

28. Updates to Chapter 36, Volume 3D

Change bars and text show changes to Chapter 36 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- Added ERETS and ERETU instructions to Far Transfers COFI Type in Table 36-1, "COFI Type for Branch Instructions".
- Added ERETS and ERETU to event handler returns in Section 36.2.4, "Event Tracing".
- Added ERET in Section 36.2.7.2, "Table of Physical Addresses (ToPA)" under "ToPA PMI".
- Updated Table 36-22, "FUP Packet Definition" in Section 36.4.2.6, "Flow Update (FUP) Packet" by adding FRED transitions to the general scenario.
- Added clarification to Software Interrupt Event in Table 36-23, "FUP Cases and IP Payload".
- Updated Table 36-49, "Control Flow Event Packet Definition" in Section 36.4.2.29, "Control Flow Event (CFE) Packet" by adding FRED transitions to the general scenario.
- Updated Table 36-50, "CFE Packet Type and Vector Fields Details" to include information for FRED in the INTR and IRET CFE Subtype and add SWINTR and SYSCALL CFE Subtypes.
- Updated case 26 (UINTR) and 27 (UIRET) in Table 36-55, "Packet Generation under Different Example Operations".
- Added cases 23 and 24 for FRED to Table 36-59, "Event Trace Examples when TriggerEn && ContextEn && EventEn is True".

CHAPTER 36

INTEL® PROCESSOR TRACE

36.1 OVERVIEW

Intel® Processor Trace (**Intel PT**) is an extension of Intel® Architecture that captures information about software execution using dedicated hardware facilities that cause only minimal performance perturbation to the software being traced. This information is collected in **data packets**. The initial implementations of Intel PT offer **control flow tracing**, which generates a variety of packets to be processed by a software decoder. The packets include timing, program flow information (e.g., branch targets, branch taken/not taken indications) and program-induced mode related information (e.g., Intel TSX state transitions, CR3 changes). These packets may be buffered internally before being sent to the memory subsystem or other output mechanism available in the platform. Debug software can process the trace data and reconstruct the program flow.

Intel Processor Trace was first introduced in Intel® processors based on Broadwell microarchitecture and Intel Atom® processors based on Goldmont microarchitecture. Later generations include additional trace sources, including software trace instrumentation using PTWRITE, and Power Event tracing.

36.1.1 Features and Capabilities

Intel PT's control flow trace generates a variety of packets that, when combined with the binaries of a program by a post-processing tool, can be used to produce an exact execution trace. The packets record flow information such as instruction pointers (IP), indirect branch targets, and directions of conditional branches within contiguous code regions (basic blocks).

Intel PT can also be configured to log software-generated packets using PTWRITE, and packets describing processor power management events. Further, Precise Event-Based Sampling (PEBS) can be configured to log PEBS records in the Intel PT trace; see Section 22.5.5.2.

In addition, the packets record other contextual, timing, and bookkeeping information that enables both functional and performance debugging of applications. Intel PT has several control and filtering capabilities available to customize the tracing information collected and to append other processor state and timing information to enable debugging. For example, there are modes that allow packets to be filtered based on the current privilege level (CPL) or the value of CR3.

Configuration of the packet generation and filtering capabilities are programmed via a set of MSRs. The MSRs generally follow the naming convention of IA32_RTIT_*. The capability provided by these configuration MSRs are enumerated by CPUID, see Section 36.3. Details of the MSRs for configuring Intel PT are described in Section 36.2.8.

36.1.1.1 Packet Summary

After a tracing tool has enabled and configured the appropriate MSRs, the processor will collect and generate trace information in the following categories of packets (for more details on the packets, see Section 36.4):

- Packets about basic information on program execution; these include:
 - Packet Stream Boundary (PSB) packets: PSB packets act as 'heartbeats' that are generated at regular intervals (e.g., every 4K trace packet bytes). These packets allow the packet decoder to find the packet boundaries within the output data stream; a PSB packet should be the first packet that a decoder looks for when beginning to decode a trace.
 - Paging Information Packet (PIP): PIPs record modifications made to the CR3 register. This information, along with information from the operating system on the CR3 value of each process, allows the debugger to attribute linear addresses to their correct application source.
 - Time-Stamp Counter (TSC) packets: TSC packets aid in tracking wall-clock time, and contain some portion of the software-visible time-stamp counter.
 - Core Bus Ratio (CBR) packets: CBR packets contain the core:bus clock ratio.

- Mini Time Counter (MTC) packets: MTC packets provide periodic indication of the passing of wall-clock time.
- Cycle Count (CYC) packets: CYC packets provide indication of the number of processor core clock cycles that pass between packets.
- Overflow (OVF) packets: OVF packets are sent when the processor experiences an internal buffer overflow, resulting in packets being dropped. This packet notifies the decoder of the loss and can help the decoder to respond to this situation.
- Packets about control flow information:
 - Taken Not-Taken (TNT) packets: TNT packets track the “direction” of direct conditional branches (taken or not taken).
 - Target IP (TIP) packets: TIP packets record the target IP of indirect branches, exceptions, interrupts, and other branches or events. These packets can contain the IP, although that IP value may be compressed by eliminating upper bytes that match the last IP. There are various types of TIP packets; they are covered in more detail in Section 36.4.2.2.
 - Flow Update Packets (FUP): FUPs provide the source IP addresses for asynchronous events (interrupt and exceptions), as well as other cases where the source address cannot be determined from the binary.
 - MODE packets: These packets provide the decoder with important processor execution information so that it can properly interpret the dis-assembled binary and trace log. MODE packets have a variety of formats that indicate details such as the execution mode (16-bit, 32-bit, or 64-bit).
- Packets inserted by software:
 - PTWRITE (PTW) packets: includes the value of the operand passed to the PTWRITE instruction (see “PTWRITE—Write Data to a Processor Trace Packet” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B).
- Packets about processor power management events:
 - MWAIT packets: Indicate successful completion of an MWAIT operation to a C-state deeper than C0.0.
 - Power State Entry (PWRE) packets: Indicate entry to a C-state deeper than C0.0.
 - Power State Exit (PWRX) packets: Indicate exit from a C-state deeper than C0.0, returning to C0.
 - Execution Stopped (EXSTOP) packets: Indicate that software execution has stopped, due to events such as P-state change, C-state change, or thermal throttling.
- Packets containing groups of processor state values:
 - Block Begin Packets (BBP): Indicate the type of state held in the following group.
 - Block Item Packets (BIP): Indicate the state values held in the group.
 - Block End Packets (BEP): Indicate the end of the current group.

36.2 INTEL® PROCESSOR TRACE OPERATIONAL MODEL

This section describes the overall Intel Processor Trace mechanism and the essential concepts relevant to how it operates.

36.2.1 Change of Flow Instruction (COFI) Tracing

A basic program block is a section of code where no jumps or branches occur. The instruction pointers (IPs) in this block of code need not be traced, as the processor will execute them from start to end without redirecting code flow. Instructions such as branches, and events such as exceptions or interrupts, can change the program flow. These instructions and events that change program flow are called Change of Flow Instructions (COFI). There are three categories of COFI:

- Direct transfer COFI.
- Indirect transfer COFI.
- Far transfer COFI.

The following subsections describe the COFI events that result in trace packet generation. Table 36-1 lists branch instruction by COFI types. For detailed description of specific instructions, see the Intel® 64 and IA-32 Architectures Software Developer's Manual.

Table 36-1. COFI Type for Branch Instructions

COFI Type	Instructions
Conditional Branch	JA, JAE, JB, JBE, JC, JCXZ, JECXZ, JRCXZ, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ, LOOP, LOOPE, LOOPNE, LOOPNZ, LOOPZ
Unconditional Direct Branch	JMP (E9 xx, EB xx), CALL (E8 xx)
Indirect Branch	JMP (FF /4), CALL (FF /2), RET (C3, C2 xx)
Far Transfers	ERETS, ERETU, INT1, INT3, INT <i>n</i> , INTO, IRET, IRETD, IRETQ, JMP (EA xx, FF /5), CALL (9A xx, FF /3), RET (CB, CA xx), SYSCALL, SYSRET, SYSENTER, SYSEXIT, VMLAUNCH, VMRESUME

36.2.1.1 Direct Transfer COFI

Direct Transfer COFI are relative branches. This means that their target is an IP whose offset from the current IP is embedded in the instruction bytes. It is not necessary to indicate target of these instructions in the trace output since it can be obtained through the source disassembly. Conditional branches need to indicate only whether the branch is taken or not. Unconditional branches do not need any recording in the trace output. There are two sub-categories:

- **Conditional Branch (Jcc, J*CXZ) and LOOP**

To track this type of instruction, the processor encodes a single bit (taken or not taken — TNT) to indicate the program flow after the instruction.

Jcc, J*CXZ, and LOOP can be traced with TNT bits. To improve the trace packet output efficiency, the processor will compact several TNT bits into a single packet.

- **Unconditional Direct Jumps**

There is no trace output required for direct unconditional jumps (like JMP near relative or CALL near relative) since they can be directly inferred from the application assembly. Direct unconditional jumps do not generate a TNT bit or a Target IP packet, though TIP.PGD and TIP.PGE packets can be generated by unconditional direct jumps that toggle Intel PT enables (see Section 36.2.6).

36.2.1.2 Indirect Transfer COFI

Indirect transfer instructions involve updating the IP from a register or memory location. Since the register or memory contents can vary at any time during execution, there is no way to know the target of the indirect transfer until the register or memory contents are read. As a result, the disassembled code is not sufficient to determine the target of this type of COFI. Therefore, tracing hardware must send out the destination IP in the trace packet for debug software to determine the target address of the COFI. Note that this IP may be a linear or effective address (see Section 36.3.1.1).

An indirect transfer instruction generates a Target IP Packet (TIP) that contains the target address of the branch. There are two sub-categories:

- **Near JMP Indirect and Near Call Indirect**

As previously mentioned, the target of an indirect COFI resides in the contents of either a register or memory location. Therefore, the processor must generate a packet that includes this target address to allow the decoder to determine the program flow.

- **Near RET**

When a CALL instruction executes, it pushes onto the stack the address of the next instruction following the CALL. Upon completion of the call procedure, the RET instruction is often used to pop the return address off of the call stack and redirect code flow back to the instruction following the CALL.

A RET instruction simply transfers program flow to the address it popped off the stack. Because a called procedure may change the return address on the stack before executing the RET instruction, debug software

can be misled if it assumes that code flow will return to the instruction following the last CALL. Therefore, even for near RET, a Target IP Packet may be sent.

— RET Compression

A special case is applied if the target of the RET is consistent with what would be expected from tracking the CALL stack. If it is assured that the decoder has seen the corresponding CALL (with “corresponding” defined as the CALL with matching stack depth), and the RET target is the instruction after that CALL, the RET target may be “compressed”. In this case, only a single TNT bit of “taken” is generated instead of a Target IP Packet. To ensure that the decoder will not be confused in cases of RET compression, only RETs that correspond to CALLs which have been seen since the last PSB packet may be compressed in a given logical processor. For details, see “Indirect Transfer Compression for Returns (RET)” in Section 36.4.2.2.

36.2.1.3 Far Transfer COFI

All operations that change the instruction pointer and are not near jumps are “far transfers”. This includes exceptions, interrupts, traps, TSX aborts, and instructions that do far transfers.

All far transfers will produce a Target IP (TIP) packet, which provides the destination IP address. For those far transfers that cannot be inferred from the binary source (e.g., asynchronous events such as exceptions and interrupts), the TIP will be preceded by a Flow Update packet (FUP), which provides the source IP address at which the event was taken. Table 36-23 indicates exactly which IP will be included in the FUP generated by a far transfer.

36.2.2 Software Trace Instrumentation with PTWRITE

PTWRITE provides a mechanism by which software can instrument the Intel PT trace. PTWRITE is a ring3-accessible instruction that can be passed to a register or memory variable, see “PTWRITE—Write Data to a Processor Trace Packet” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B, for details. The contents of that variable will be used as the payload for the PTW packet (see Table 36-40 “PTW Packet Definition”), inserted at the time of PTWRITE retirement, assuming PTWRITE is enabled and all other filtering conditions are met. Decode and analysis software will then be able to determine the meaning of the PTWRITE packet based on the IP of the associated PTWRITE instruction.

PTWRITE is enabled via IA32_RTIT_CTL.PTWEn[12] (see Table 36-6). Optionally, the user can use IA32_RTIT_CTL.FUPonPTW[5] to enable PTW packets to be followed by FUP packets containing the IP of the associated PTWRITE instruction. Support for PTWRITE is introduced in Intel Atom processors based on the Goldmont Plus microarchitecture.

36.2.3 Power Event Tracing

Power Event Trace is a capability that exposes core- and thread-level sleep state and power down transition information. When this capability is enabled, the trace will expose information about:

- Scenarios where software execution stops.
 - Due to sleep state entry, frequency change, or other powerdown.
 - Includes the IP, when in the tracing context.
- The requested and resolved hardware thread C-state.
 - Including indication of hardware autonomous C-state entry.
- The last and deepest core C-state achieved during a sleep session.
- The reason for C-state wake.

This information is in addition to the bus ratio (CBR) information provided by default after any powerdown, and the timing information (TSC, TMA, MTC, CYC) provided during or after a powerdown state.

Power Event Trace is enabled via IA32_RTIT_CTL.PwrEvtEn[4]. Support for Power Event Tracing is introduced in Intel Atom processors based on the Goldmont Plus microarchitecture.

36.2.4 Event Tracing

Event Trace is a capability that exposes details about the asynchronous events, when they are generated, and when their corresponding software event handler completes execution. These include:

- Interrupts, including NMI and SMI, including the interrupt vector when defined.
- Faults, exceptions including the fault vector.
 - Page faults additionally include the page fault address, when in context.
- Event handler returns, including `ERETS`, `ERETU`, `IRET`, and `RSM`.
- VM exits and VM entries.¹
 - VM exits include the values written to the “exit reason” and “exit qualification” VMCS fields.
- INIT and SIPI events.
- TSX aborts, including the abort status returned for the RTM instructions.
- Shutdown.

Additionally, it provides indication of the status of the Interrupt Flag (IF), to indicate when interrupts are masked.

Event Trace is enabled via `IA32_RTIT_CTL.EventEn[31]`. Event Trace information is conveyed in Control Flow Event (CFE) and Event Data (EVD) packets, as well as the legacy `MODE.Exec` packet. See Section 36.4.2 for packet details. Support for Event Trace is introduced in Intel® processors based on Gracemont microarchitecture.

36.2.5 Trace Filtering

Intel Processor Trace provides filtering capabilities, by which the debug/profile tool can control what code is traced.

36.2.5.1 Filtering by Current Privilege Level (CPL)

Intel PT provides the ability to configure a logical processor to generate trace packets only when `CPL = 0`, when `CPL > 0`, or regardless of `CPL`.

CPL filtering ensures that no IPs or other architectural state information associated with the filtered `CPL` can be seen in the log. For example, if the processor is configured to trace only when `CPL > 0`, and software executes `SYSCALL` (changing the `CPL` to 0), the destination IP of the `SYSCALL` will be suppressed from the generated packet (see the discussion of `TIP.PGD` in Section 36.4.2.5).

It should be noted that `CPL` is always 0 in real-address mode and that `CPL` is always 3 in virtual-8086 mode. To trace code in these modes, filtering should be configured accordingly.

When software is executing in a non-enabled `CPL`, `ContextEn` is cleared. See Section 36.2.6.1 for details.

36.2.5.2 Filtering by CR3

Intel PT supports a `CR3`-filtering mechanism by which the generation of packets containing architectural states can be enabled or disabled based on the value of `CR3`. A debugger can use `CR3` filtering to trace only a single application without context switching the state of the `RTIT` MSRs. For the reconstruction of traces from software with multiple threads, debug software may wish to context-switch for the state of the `RTIT` MSRs (if the operating system does not provide context-switch support) to separate the output for the different threads (see Section 36.3.5, “Context Switch Consideration”).

To trace for only a single `CR3` value, software can write that value to the `IA32_RTIT_CR3_MATCH` MSR, and set `IA32_RTIT_CTL.CR3Filter`. When `CR3` value does not match `IA32_RTIT_CR3_MATCH` and `IA32_RTIT_CTL.CR3Filter` is 1, `ContextEn` is forced to 0, and packets containing architectural states will not be generated. Some other packets can be generated when `ContextEn` is 0; see Section 36.2.6.3 for details. When `CR3` does match `IA32_RTIT_CR3_MATCH` (or when `IA32_RTIT_CTL.CR3Filter` is 0), `CR3` filtering does not force `ContextEn` to 0 (although it could be 0 due to other filters or modes).

1. Logging of VMX transitions depends on VMCS configuration, see Section 36.5.1.

CR3 matches IA32_RTIT_CR3_MATCH if the two registers are identical for bits 63:12, or 63:5 when in PAE paging mode; the lower 5 bits of CR3 and IA32_RTIT_CR3_MATCH are ignored. CR3 filtering is independent of the value of CR0.PG.

When CR3 filtering is in use, PIP packets may still be seen in the log if the processor is configured to trace when CPL = 0 (IA32_RTIT_CTL.OS = 1). If not, no PIP packets will be seen.

36.2.5.3 Filtering by IP

Trace packet generation with configurable filtering by IP is supported if CPUID.14H.00H:EBX[2] = 1. Intel PT can be configured to enable the generation of packets containing architectural states only when the processor is executing code within certain IP ranges. If the IP is outside of these ranges, generation of some packets is blocked.

IP filtering is enabled using the ADDRn_CFG fields in the IA32_RTIT_CTL MSR (Section 36.2.8.2), where the digit 'n' is a zero-based number that selects which address range is being configured. Each ADDRn_CFG field configures the use of the register pair IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B (Section 36.2.8.5). IA32_RTIT_ADDRn_A defines the base and IA32_RTIT_ADDRn_B specifies the limit of the range in which tracing is enabled. Thus each range, referred to as the ADDRn range, is defined by [IA32_RTIT_ADDRn_A, IA32_RTIT_ADDRn_B]. There can be multiple such ranges, software can query CPUID (Section 36.3.1) for the number of ranges supported on a processor.

Default behavior (ADDRn_CFG=0) defines no IP filter range, meaning FilterEn is always set. In this case code at any IP can be traced, though other filters, such as CR3 or CPL, could limit tracing. When ADDRn_CFG is set to enable IP filtering (see Section 36.3.1), tracing will commence when a taken branch or event is seen whose target address is in the ADDRn range.

While inside a tracing region and with FilterEn is set, leaving the tracing region may only be detected once a taken branch or event with a target outside the range is retired. If an ADDRn range is entered or exited by executing the next sequential instruction, rather than by a control flow transfer, FilterEn may not toggle immediately. See Section 36.2.6.5 for more details on FilterEn.

Note that these address range base and limit values are inclusive, such that the range includes the first and last instruction whose first instruction byte is in the ADDRn range.

Depending upon processor implementation, IP filtering may be based on linear or effective address. This can cause different behavior between implementations if CSbase is not equal to zero or in real mode. See Section 36.3.1.1 for details. Software can query CPUID to determine filters are based on linear or effective address (Section 36.3.1).

Note that some packets, such as MTC (Section 36.3.7) and other timing packets, do not depend on FilterEn. For details on which packets depend on FilterEn, and hence are impacted by IP filtering, see Section 36.4.1.

TraceStop

The ADDRn ranges can also be configured to cause tracing to be disabled upon entry to the specified region. This is intended for cases where unexpected code is executed, and the user wishes to immediately stop generating packets in order to avoid overwriting previously written packets.

The TraceStop mechanism works much the same way that IP filtering does, and uses the same address comparison logic. The TraceStop region base and limit values are programmed into one or more ADDRn ranges, but IA32_RTIT_CTL.ADDRn_CFG is configured with the TraceStop encoding. Like FilterEn, TraceStop is detected when a taken branch or event lands in a TraceStop region.

Further, TraceStop requires that TriggerEn=1 at the beginning of the branch/event, and ContextEn=1 upon completion of the branch/event. When this happens, the CPU will set IA32_RTIT_STATUS.Stopped, thereby clearing TriggerEn and hence disabling packet generation. This may generate a TIP.PGD packet with the target IP of the branch or event that entered the TraceStop region. Finally, a TraceStop packet will be inserted, to indicate that the condition was hit.

If a TraceStop condition is encountered during buffer overflow (Section 36.3.8), it will not be dropped, but will instead be signaled once the overflow has resolved.

Note that a TraceStop event does not guarantee that all internally buffered packets are flushed out of internal buffers. To ensure that this has occurred, the user should clear TraceEn.

To resume tracing after a TraceStop event, the user must first disable Intel PT by clearing IA32_RTIT_CTL.TraceEn before the IA32_RTIT_STATUS.Stopped bit can be cleared. At this point Intel PT can be reconfigured, and tracing resumed.

Note that the IA32_RTIT_STATUS.Stopped bit can also be set using the ToPA STOP bit. See Section 36.2.7.2.

IP Filtering Example

The following table gives an example of IP filtering behavior. Assume that IA32_RTIT_ADDRn_A = the IP of RangeBase, and that IA32_RTIT_ADDRn_B = the IP of RangeLimit, while IA32_RTIT_CTL.ADDRn_CFG = 0x1 (enable ADDRn range as a FilterEn range).

Table 36-2. IP Filtering Packet Example

Code Flow	Packets
<pre> Bar: jmp RangeBase // jump into filter range RangeBase: jcc Foo // not taken add eax, 1 Foo: jmp RangeLimit+1 // jump out of filter range RangeLimit: nop jcc Bar </pre>	<pre> TIP.PGE(RangeBase) TNT(0) TIP.PGD(RangeLimit+1) </pre>

IP Filtering and TraceStop

It is possible for the user to configure IP filter range(s) and TraceStop range(s) that overlap. In this case, code executing in the non-overlapping portion of either range will behave as would be expected from that range. Code executing in the overlapping range will get TraceStop behavior.

36.2.6 Packet Generation Enable Controls

Intel Processor Trace includes a variety of controls that determine whether a packet is generated. In general, most packets are sent only if Packet Enable (**PacketEn**) is set. PacketEn is an internal state maintained in hardware in response to software configurable enable controls, PacketEn is not visible to software directly. The relationship of PacketEn to the software-visible controls in the configuration MSRs is described in this section.

36.2.6.1 Packet Enable (PacketEn)

When PacketEn is set, the processor is in the mode that Intel PT is monitoring. PacketEn is composed of other states according to this relationship:

```
PacketEn := TriggerEn AND ContextEn AND FilterEn AND BranchEn
```

These constituent controls are detailed in the following subsections.

PacketEn ultimately determines when the processor is tracing. When PacketEn is set, all control flow packets are enabled. When PacketEn is clear, no control flow packets are generated, though other packets (timing and book-keeping packets) may still be sent. See Section 36.2.7 for details of PacketEn and packet generation.

Note that, on processors that do not support IP filtering (i.e., CPUID.14H.00H:EBX[2] = 0), FilterEn is treated as always set.

36.2.6.2 Trigger Enable (TriggerEn)

Trigger Enable (**TriggerEn**) is the primary indicator that trace packet generation is active. TriggerEn is set when IA32_RTIT_CTL.TraceEn is set, and cleared by any of the following conditions:

- TraceEn is cleared by software.

- A TraceStop condition is encountered and IA32_RTIT_STATUS.Stopped is set.
- IA32_RTIT_STATUS.Error is set due to an operational error (see Section 36.3.10).

Software can discover the current TriggerEn value by reading the IA32_RTIT_STATUS.TriggerEn bit. When TriggerEn is clear, tracing is inactive and no packets are generated.

36.2.6.3 Context Enable (ContextEn)

Context Enable (**ContextEn**) indicates whether the processor is in the state or mode that software configured hardware to trace. For example, if execution with CPL = 0 code is not being traced (IA32_RTIT_CTL.OS = 0), then ContextEn will be 0 when the processor is in CPL0.

Software can discover the current ContextEn value by reading the IA32_RTIT_STATUS.ContextEn bit. ContextEn is defined as follows:

```
ContextEn = !((IA32_RTIT_CTL.OS = 0 AND CPL = 0) OR
(IA32_RTIT_CTL.USER = 0 AND CPL > 0) OR (IS_IN_A_PRODUCTION_ENCLAVE1) OR
(IA32_RTIT_CTL.CR3Filter = 1 AND IA32_RTIT_CR3_MATCH does not match CR3))
```

If the clearing of ContextEn causes PacketEn to be cleared, a Packet Generation Disable (TIP.PGD) packet is generated, but its IP payload is suppressed. If the setting of ContextEn causes PacketEn to be set, a Packet Generation Enable (TIP.PGE) packet is generated.

When ContextEn is 0, control flow packets (TNT, FUP, TIP.*, MODE.*) are not generated, and no Linear Instruction Pointers (LIPs) are exposed. However, some packets, such as MTC and PSB (see Section 36.4.2.16 and Section 36.4.2.17), may still be generated while ContextEn is 0. For details of which packets are generated only when ContextEn is set, see Section 36.4.1.

The processor does not update ContextEn when TriggerEn = 0.

The value of ContextEn will toggle only when TriggerEn = 1.

36.2.6.4 Branch Enable (BranchEn)

This value is based purely on the IA32_RTIT_CTL.BranchEn value. If **BranchEn** is not set, then relevant COFI packets (TNT, TIP*, FUP, MODE.*) are suppressed. Other packets related to timing (TSC, TMA, MTC, CYC), as well as PSB, will be generated normally regardless. Further, PIP and VMCS continue to be generated, as indicators of what software is running.

36.2.6.5 Filter Enable (FilterEn)

Filter Enable indicates that the Instruction Pointer (IP) is within the range of IPs that Intel PT is configured to watch. Software can get the state of Filter Enable by a RDMSR of IA32_RTIT_STATUS.FilterEn. For details on configuration and use of IP filtering, see Section 36.2.5.3.

On clearing of FilterEn that also clears PacketEn, a Packet Generation Disable (TIP.PGD) will be generated, but unlike the ContextEn case, the IP payload may not be suppressed. For direct, unconditional branches, as well as for indirect branches (including RETs), the PGD generated by leaving the tracing region and clearing FilterEn will contain the target IP. This means that IPs from outside the configured range can be exposed in the trace, as long as they are within context.

When FilterEn is 0, control flow packets are not generated (e.g., TNT, TIP). However, some packets, such as PIP, MTC, and PSB, may still be generated while FilterEn is clear. For details on packet enable dependencies, see Section 36.4.1.

After TraceEn is set, FilterEn is set to 1 at all times if there is no IP filter range configured by software (IA32_RTIT_CTL.ADDRn_CFG != 1, for all n), or if the processor does not support IP filtering (i.e., CPUID.14H.00H:EBX[2] = 0). FilterEn will toggle only when TraceEn=1 and ContextEn=1, and when at least one range is configured for IP filtering.

1. Trace packets generation is disabled in a production enclave, see Section 36.2.9.5. See the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D, about differences between a production enclave and a debug enclave.

36.2.7 Trace Output

Intel PT output should be viewed independently from trace content and filtering mechanisms. The options available for trace output can vary across processor generations and platforms.

Trace output is written out using one of the following output schemes, as configured by the ToPA and FabricEn bit fields of IA32_RTIT_CTL (see Section 36.2.8.2):

- A single, contiguous region of physical address space.
- A collection of variable-sized regions of physical memory. These regions are linked together by tables of pointers to those regions, referred to as Table of Physical Addresses (**ToPA**). The trace output stores bypass the caches and the TLBs, but are not serializing. This is intended to minimize the performance impact of the output.
- A platform-specific trace transport subsystem.

Regardless of the output scheme chosen, Intel PT stores bypass the processor caches by default. This ensures that they don't consume precious cache space, but they do not have the serializing aspects associated with un-cacheable (UC) stores. Software should avoid using MTRRs to mark any portion of the Intel PT output region as UC, as this may override the behavior described above and force Intel PT stores to UC, thereby incurring severe performance impact.

There is no guarantee that a packet will be written to memory or other trace endpoint after some fixed number of cycles after a packet-producing instruction executes. The only way to assure that all packets generated have reached their endpoint is to clear TraceEn and follow that with a store, fence, or serializing instruction; doing so ensures that all buffered packets are flushed out of the processor.

36.2.7.1 Single Range Output

When IA32_RTIT_CTL.ToPA and IA32_RTIT_CTL.FabricEn bits are clear, trace packet output is sent to a single, contiguous memory (or MMIO if DRAM is not available) range defined by a base address in IA32_RTIT_OUTPUT_BASE (Section 36.2.8.7) and mask value in IA32_RTIT_OUTPUT_MASK_PTRS (Section 36.2.8.8). The current write pointer in this range is also stored in IA32_RTIT_OUTPUT_MASK_PTRS. This output range is circular, meaning that when the writes wrap around the end of the buffer they begin again at the base address.

This output method is best suited for cases where Intel PT output is either:

- Configured to be directed to a sufficiently large contiguous region of DRAM.
- Configured to go to an MMIO debug port, in order to route Intel PT output to a platform-specific trace endpoint (e.g., JTAG). In this scenario, a specific range of addresses is written in a circular manner, and SoC will intercept these writes and direct them to the proper device. Repeated writes to the same address do not overwrite each other, but are accumulated by the debugger, and hence no data is lost by the circular nature of the buffer.

The processor will determine the address to which to write the next trace packet output byte as follows:

```
OutputBase[63:0] := IA32_RTIT_OUTPUT_BASE[63:0]
OutputMask[63:0] := ZeroExtend64(IA32_RTIT_OUTPUT_MASK_PTRS[31:0])
OutputOffset[63:0] := ZeroExtend64(IA32_RTIT_OUTPUT_MASK_PTRS[63:32])
trace_store_phys_addr := (OutputBase & ~OutputMask) + (OutputOffset & OutputMask)
```

Single-Range Output Errors

If the output base and mask are not properly configured by software, an operational error (see Section 36.3.10) will be signaled, and tracing disabled. Error scenarios with single-range output are:

- Mask value is non-contiguous.
IA32_RTIT_OUTPUT_MASK_PTRS.MaskOrTablePointer value has a 0 in a less significant bit position than the most significant bit containing a 1.
- Base address and Mask are mis-aligned, and have overlapping bits set.
IA32_RTIT_OUTPUT_BASE && IA32_RTIT_OUTPUT_MASK_PTRS[31:0] > 0.
- Illegal Output Offset
IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset is greater than the mask value IA32_RTIT_OUTPUT_MASK_PTRS[31:0].

Also note that errors can be signaled due to trace packet output overlapping with restricted memory, see Section 36.2.7.4.

36.2.7.2 Table of Physical Addresses (ToPA)

When IA32_RTIT_CTL.ToPA is set and IA32_RTIT_CTL.FabricEn is clear, the ToPA output mechanism is utilized. The ToPA mechanism uses a linked list of tables; see Figure 36-1 for an illustrative example. Each entry in the table contains some attribute bits, a pointer to an output region, and the size of the region. The last entry in the table may hold a pointer to the next table. This pointer can either point to the top of the current table (for circular array) or to the base of another table. The table size is not fixed, since the link to the next table can exist at any entry.

The processor treats the various output regions referenced by the ToPA table(s) as a unified buffer. This means that a single packet may span the boundary between one output region and the next.

The ToPA mechanism is controlled by three values maintained by the processor:

- **proc_trace_table_base.**
This is the physical address of the base of the current ToPA table. When tracing is enabled, the processor loads this value from the IA32_RTIT_OUTPUT_BASE MSR. While tracing is enabled, the processor updates the IA32_RTIT_OUTPUT_BASE MSR with changes to proc_trace_table_base, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_table_base.
- **proc_trace_table_offset.**
This indicates the entry of the current table that is currently in use. (This entry contains the address of the current output region.) When tracing is enabled, the processor loads the value from bits 31:7 (MaskOrTableOffset) of the IA32_RTIT_OUTPUT_MASK_PTRS into bits 27:3 of proc_trace_table_offset. While tracing is enabled, the processor updates IA32_RTIT_OUTPUT_MASK_PTRS.MaskOrTableOffset with changes to proc_trace_table_offset, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_table_offset.
- **proc_trace_output_offset.**
This is a pointer into the current output region and indicates the location of the next write. When tracing is enabled, the processor loads this value from bits 63:32 (OutputOffset) of the IA32_RTIT_OUTPUT_MASK_PTRS. While tracing is enabled, the processor updates IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset with changes to proc_trace_output_offset, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_output_offset.

Figure 36-1 provides an illustration (not to scale) of the table and associated pointers.

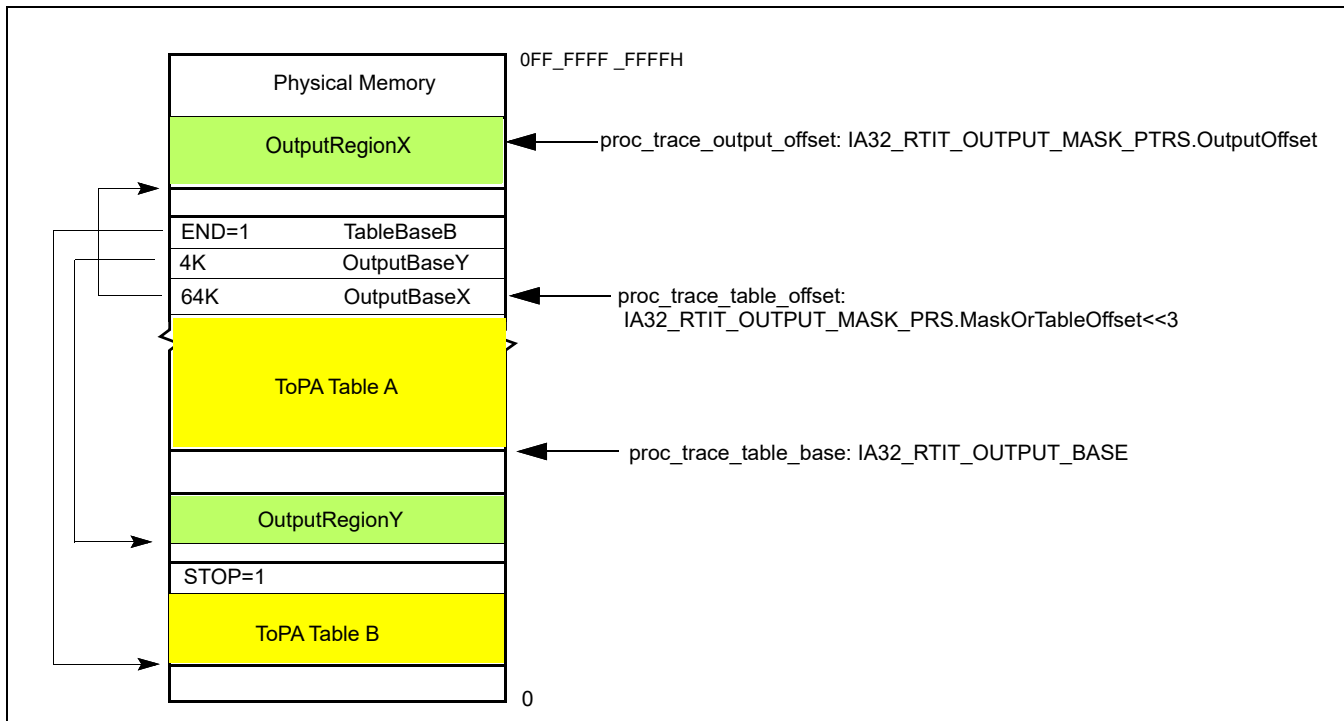


Figure 36-1. ToPA Memory Illustration

With the ToPA mechanism, the processor writes packets to the current output region (identified by `proc_trace_table_base` and the `proc_trace_table_offset`). The offset within that region to which the next byte will be written is identified by `proc_trace_output_offset`. When that region is filled with packet output (thus `proc_trace_output_offset = RegionSize-1`), `proc_trace_table_offset` is moved to the next ToPA entry, `proc_trace_output_offset` is set to 0, and packet writes begin filling the new output region specified by `proc_trace_table_offset`.

As packets are written out, each store derives its physical address as follows:

```
trace_store_phys_addr := Base address from current ToPA table entry +
proc_trace_output_offset
```

Eventually, the regions represented by all entries in the table may become full, and the final entry of the table is reached. An entry can be identified as the final entry because it has either the END or STOP attribute. The END attribute indicates that the address in the entry does not point to another output region, but rather to another ToPA table. The STOP attribute indicates that tracing will be disabled once the corresponding region is filled. See Table 36-3 and the section that follows for details on STOP.

When an END entry is reached, the processor loads `proc_trace_table_base` with the base address held in this END entry, thereby moving the current table pointer to this new table. The `proc_trace_table_offset` is reset to 0, as is the `proc_trace_output_offset`, and packet writes will resume at the base address indicated in the first entry.

If the table has no STOP or END entry, and trace-packet generation remains enabled, eventually the maximum table size will be reached (`proc_trace_table_offset = 0FFFFFF8H`). In this case, the `proc_trace_table_offset` and `proc_trace_output_offset` are reset to 0 (wrapping back to the beginning of the current table) once the last output region is filled.

It is important to note that processor updates to the IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs are asynchronous to instruction execution. Thus, reads of these MSRs while Intel PT is enabled may return stale values. Like all IA32_RTIT_* MSRs, the values of these MSRs should not be trusted or saved unless trace packet generation is first disabled by clearing IA32_RTIT_CTL.TraceEn. This ensures that the output MSR values account for all packets generated to that point, after which the processor will cease updating the output MSR values until tracing resumes.¹

The processor may cache internally any number of entries from the current table or from tables that it references (directly or indirectly). If tracing is enabled, the processor may ignore or delay detection of modifications to these tables. To ensure that table changes are detected by the processor in a predictable manner, software should clear TraceEn before modifying the current table (or tables that it references) and only then re-enable packet generation.

Single Output Region ToPA Implementation

The first processor generation to implement Intel PT supports only ToPA configurations with a single ToPA entry followed by an END entry that points back to the first entry (creating one circular output buffer). Such processors enumerate CPUID.14H.00H:ECX.MENTRY[1] = 0 and CPUID.14H.00H:ECX.TOPAOUT[0] = 1.

If CPUID.14H.00H:ECX.MENTRY[1] = 0, ToPA tables can hold only one output entry, which must be followed by an END=1 entry which points back to the base of the table. Hence only one contiguous block can be used as output.

The lone output entry can have INT or STOP set, but nonetheless must be followed by an END entry as described above. Note that, if INT=1, the PMI will actually be delivered before the region is filled.

ToPA Table Entry Format

The format of ToPA table entries is shown in Figure 36-2. The size of the address field is determined by the processor’s physical-address width MAXPHYADDR.²

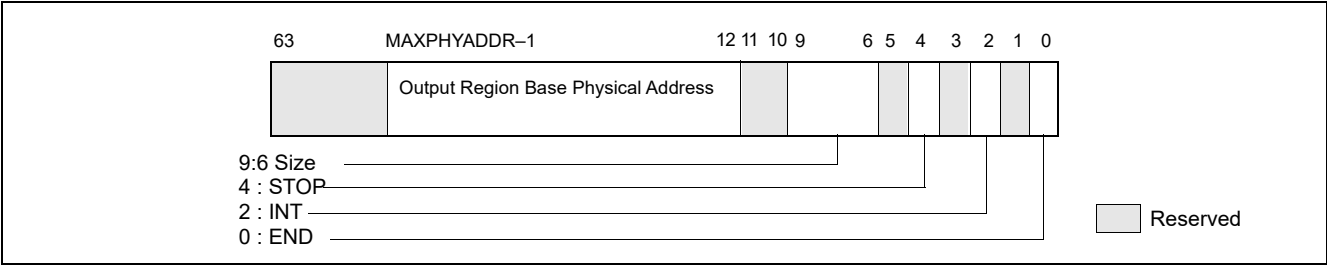


Figure 36-2. Layout of ToPA Table Entry

Table 36-3 describes the details of the ToPA table entry fields. If reserved bits are set to 1, an error is signaled.

1. Although WRMSR is a serializing instruction, the execution of WRMSR that forces packet writes by clearing TraceEn does not itself cause these writes to be globally observed.

2. Usually, MAXPHYADDR is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

Table 36-3. ToPA Table Entry Fields

ToPA Entry Field	Description
Output Region Base Physical Address	If END=0, this is the base physical address of the output region specified by this entry. Note that all regions must be aligned based on their size. Thus a 2M region must have bits 20:12 clear. If the region is not properly aligned, an operational error will be signaled when the entry is reached. If END=1, this is the 4K-aligned base physical address of the next ToPA table (which may be the base of the current table, or the first table in the linked list if a circular buffer is desired). If the processor supports only a single ToPA output region (see above), this address must be the value currently in the IA32_RTIT_OUTPUT_BASE MSR.
Size	Indicates the size of the associated output region. Encodings are: 0: 4K, 1: 8K, 2: 16K, 3: 32K, 4: 64K, 5: 128K, 6: 256K, 7: 512K, 8: 1M, 9: 2M, 10: 4M, 11: 8M, 12: 16M, 13: 32M, 14: 64M, 15: 128M This field is ignored if END=1.
STOP	When the output region indicated by this entry is filled, software should disable packet generation. This will be accomplished by setting IA32_RTIT_STATUS.Stopped, which clears TriggerEn. This bit must be 0 if END=1; otherwise it is treated as reserved bit violation (see ToPA Errors).
INT	When the output region indicated by this entry is filled, signal PerfMon LVT interrupt. Note that if both INT and STOP are set in the same entry, the STOP will happen before the INT. Thus the interrupt handler should expect that the IA32_RTIT_STATUS.Stopped bit will be set, and will need to be reset before tracing can be resumed. This bit must be 0 if END=1; otherwise it is treated as reserved bit violation (see ToPA Errors).
END	If set, indicates that this is an END entry, and thus the address field points to a table base rather than an output region base. If END=1, INT and STOP must be set to 0; otherwise it is treated as reserved bit violation (see ToPA Errors). The Size field is ignored in this case. If the processor supports only a single ToPA output region (see above), END must be set in the second table entry.

ToPA STOP

Each ToPA entry has a STOP bit. If this bit is set, the processor will set the IA32_RTIT_STATUS.Stopped bit when the corresponding trace output region is filled. This will clear TriggerEn and thereby cease packet generation. See Section 36.2.8.4 for details on IA32_RTIT_STATUS.Stopped. This sequence is known as “ToPA Stop”.

No TIP.PGD packet will be seen in the output when the ToPA stop occurs, since the disable happens only when the region is already full. When this occurs, output ceases after the last byte of the region is filled, which may mean that a packet is cut off in the middle. Any packets remaining in internal buffers are lost and cannot be recovered.

When ToPA stop occurs, the IA32_RTIT_OUTPUT_BASE MSR will hold the base address of the table whose entry had STOP=1. IA32_RTIT_OUTPUT_MASK_PTRS.MaskOffsetTableOffset will hold the index value for that entry, and the IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset should be set to the size of the region minus one.

Note that this means the offset pointer is pointing to the next byte after the end of the region, a configuration that would produce an operational error if the configuration remained when tracing is re-enabled with IA32_RTIT_STATUS.Stopped cleared.

ToPA PMI

Each ToPA entry has an INT bit. If this bit is set, the processor will signal a performance-monitoring interrupt (PMI) when the corresponding trace output region is filled. This interrupt is not precise, and it is thus likely that writes to the next region will occur by the time the interrupt is taken.

The following steps should be taken to configure this interrupt:

1. Enable PMI via the LVT Performance Monitor register (at MMIO offset 340H in xAPIC mode; via MSR 834H in x2APIC mode). See the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B, for more details on this register. For ToPA PMI, set all fields to 0, save for the interrupt vector, which can be selected by software.
2. Set up an interrupt handler to service the interrupt vector that a ToPA PMI can raise.

3. Set the interrupt flag by executing STI.
4. Set the INT bit in the ToPA entry of interest and enable packet generation, using the ToPA output option. Thus, TraceEn=ToPA=1 in the IA32_RTIT_CTL MSR.

Once the INT region has been filled with packet output data, the interrupt will be signaled. This PMI can be distinguished from others by checking bit 55 (Trace_ToPA_PMI) of the IA32_PERF_GLOBAL_STATUS MSR (MSR 38EH). Once the ToPA PMI handler has serviced the relevant buffer, writing 1 to bit 55 of the MSR at 390H (IA32_GLOBAL_STATUS_RESET) clears IA32_PERF_GLOBAL_STATUS.Trace_ToPA_PMI.

Intel PT is not frozen on PMI, and thus the interrupt handler will be traced (though filtering can prevent this). The Freeze_PerfMon_on_PMI and Freeze_LBRs_on_PMI settings in IA32_DEBUGCTL will be applied on ToPA PMI just as on other PMIs, and hence PerfMon counters are frozen.

Assuming the PMI handler wishes to read any buffered packets for persistent output, or wishes to modify any Intel PT MSRs, software should first disable packet generation by clearing TraceEn. This ensures that all buffered packets are written to memory and avoids tracing of the PMI handler. The configuration MSRs can then be used to determine where tracing has stopped. If packet generation is disabled by the handler, it should then be manually re-enabled before the IRET, ERETS, or ERETU if continued tracing is desired.

In rare cases, it may be possible to trigger a second ToPA PMI before the first is handled. This can happen if another ToPA region with INT=1 is filled before, or shortly after, the first PMI is taken, perhaps due to EFLAGS.IF being cleared for an extended period of time. This can manifest in two ways: either the second PMI is triggered before the first is taken, and hence only one PMI is taken, or the second is triggered after the first is taken, and thus will be taken when the handler for the first completes. Software can minimize the likelihood of the second case by clearing TraceEn at the beginning of the PMI handler. Further, it can detect such cases by then checking the Interrupt Request Register (IRR) for PMI pending, and checking the ToPA table base and off-set pointers (in IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS) to see if multiple entries with INT=1 have been filled.

PMI Preservation

In some cases a ToPA PMI may be taken after completion of an XSAVES instruction that saves Intel PT state, and in such cases any modification of Intel PT MSRs within the PMI handler will not persist when the saved Intel PT context is later restored with XRSTORS. To account for such a scenario, the PMI Preservation feature has been added. Support for this feature is indicated by CPUID.14H.00H:EBX[6].

When IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1, PMI preservation is enabled. When a ToPA region with INT=1 is filled, a PMI is pended and the new IA32_RTIT_STATUS.PendToPAPMI[7] is set to 1. If this bit is set when Intel PT is enabled, such that IA32_RTIT_CTL.TraceEn[0] transitions from 0 to 1, a ToPA PMI is pended. This behavior ensures that any ToPA PMI that is pended during XSAVES, and hence can't be properly handled, will be re-pended when the saved PT state is restored.

When this feature is enabled, the PMI handler should take the following actions:

1. Ignore ToPA PMIs that are taken when TraceEn = 0. This indicates that the PMI was pended during Intel PT disable, and the PendToPAPMI flag will ensure that the PMI is re-pended once Intel PT is re-enabled in the same context. For this reason, the PendToPAPMI bit should be left set to 1.
2. If TraceEn=1 and the PMI can be properly handled, clear the new PendToPAPMI bit. This will ensure that additional, spurious ToPA PMIs are not taken. It is required that PendToPAPMI is cleared before the PMI LVT mask is cleared in the APIC, and before any clearing of either LBRs_FROZEN or COUNTERS_FROZEN in IA32_PERF_GLOBAL_STATUS.

ToPA PMI and Single Output Region ToPA Implementation

A processor that supports only a single ToPA output region implementation (such that only one output region is supported; see above) will attempt to signal a ToPA PMI interrupt before the output wraps and overwrites the top of the buffer. To support this functionality, the PMI handler should disable packet generation as soon as possible.

Due to PMI skid, it is possible that, in rare cases, the wrap will have occurred before the PMI is delivered. Software can avoid this by setting the STOP bit in the ToPA entry (see Table 36-3); this will disable tracing once the region is filled, and no wrap will occur. This approach has the downside of disabling packet generation so that some of the instructions that led up to the PMI will not be traced. If the PMI skid is significant enough to cause the region to fill and tracing to be disabled, the PMI handler will need to clear the IA32_RTIT_STATUS.Stopped indication before tracing can resume.

ToPA PMI and XSAVES/XRSTORS State Handling

In some cases the ToPA PMI may be taken after completion of an XSAVES instruction that switches Intel PT state, and in such cases any modification of Intel PT MSRs within the PMI handler will not persist when the saved Intel PT context is later restored with XRSTORS. To account for such a scenario, it is recommended that the Intel PT output configuration be modified by altering the ToPA tables themselves, rather than the Intel PT output MSRs. On processors that support PMI preservation (CPUID.14H.00H:EBX[6] = 1), setting IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1 will ensure that a PMI that is pending at the time PT is disabled will be recorded by setting IA32_RTIT_STATUS.PendTopaPMI[7] = 1. A PMI will then be pended when the saved PT context is later restored.

Table 36-4 depicts a recommended PMI handler algorithm for managing multi-region ToPA output and handling ToPA PMIs that may arrive between XSAVES and XRSTORS, if PMI preservation is not in use. This algorithm is flexible to allow software to choose between adding entries to the current ToPA table, adding a new ToPA table, or using the current ToPA table as a circular buffer. It assumes that the ToPA entry that triggers the PMI is not the last entry in the table, which is the recommended treatment.

Table 36-4. Algorithm to Manage Intel PT ToPA PMI and XSAVES/XRSTORS

Pseudo Code Flow
<pre> IF (IA32_PERF_GLOBAL_STATUS.ToPA) Save IA32_RTIT_CTL value; IF (IA32_RTIT_CTL.TraceEN) Disable Intel PT by clearing TraceEn; FI; IF (there is space available to grow the current ToPA table) Add one or more ToPA entries after the last entry in the ToPA table; Point new ToPA entry address field(s) to new output region base(s); ELSE Modify an upcoming ToPA entry in the current table to have END=1; IF (output should transition to a new ToPA table) Point the address of the "END=1" entry of the current table to the new table base; ELSE /* Continue to use the current ToPA table, make a circular. */ Point the address of the "END=1" entry to the base of the current table; Modify the ToPA entry address fields for filled output regions to point to new, unused output regions; /* Filled regions are those with index in the range of 0 to (IA32_RTIT_MASK_PTRS.MaskOrTableOffset -1). */ FI; FI; Restore saved IA32_RTIT_CTL.value; FI; </pre>

ToPA Errors

When a malformed ToPA entry is found, an **operational error** results (see Section 36.3.10). A malformed entry can be any of the following:

- ToPA entry reserved bit violation.**
This describes cases where a bit marked as reserved in Section 36.2.7.2 above is set to 1.
- ToPA alignment violation.**
This includes cases where illegal ToPA entry base address bits are set to 1:
 - ToPA table base address is not 4KB-aligned. The table base can be from a WRMSR to IA32_RTIT_OUTPUT_BASE, or from a ToPA entry with END=1.
 - ToPA entry base address is not aligned to the ToPA entry size (e.g., a 2MB region with base address[20:12] not equal to 0), for ToPA entries with END=0.
 - ToPA entry base address sets physical address bits in the range 63:MAXPHYADDR.¹

3. Illegal ToPA Output Offset.

IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset is greater than or equal to the size of the current ToPA output region size.

4. ToPA rules violations.

These are similar to ToPA entry reserved bit violations; they are cases when a ToPA entry is encountered with illegal field combinations. They include the following:

- a. Setting the STOP or INT bit on an entry with END=1.
- b. Setting the END bit in entry 0 of a ToPA table.
- c. On processors that support only a single ToPA entry (see above), two additional illegal settings apply:
 - i) ToPA table entry 1 with END=0.
 - ii) ToPA table entry 1 with base address not matching the table base.

In all cases, the error will be logged by setting IA32_RTIT_STATUS.Error, thereby disabling tracing when the problematic ToPA entry is reached (when proc_trace_table_offset points to the entry containing the error). Any packet bytes that are internally buffered when the error is detected may be lost.

Note that operational errors may also be signaled due to attempts to access restricted memory. See Section 36.2.7.4 for details.

A tracing software have a range of flexibility using ToPA to manage the interaction of Intel PT with application buffers, see Section 36.4.2.26.

36.2.7.3 Trace Transport Subsystem

When IA32_RTIT_CTL.FabricEn is set, the IA32_RTIT_CTL.ToPA bit is ignored, and trace output is written to the trace transport subsystem. The endpoints of this transport are platform-specific, and details of configuration options should refer to the specific platform documentation. The FabricEn bit is available to be set if CPUID.14H.00H:EBX[3] = 1.

36.2.7.4 Restricted Memory Access

Packet output cannot be directed to any regions of memory that are restricted by the platform. In particular, all memory accesses on behalf of packet output are checked against the SMRR, PRMRR, and SEAMRR protected regions. If there is any overlap with these regions, trace data collection will not function properly. Exact processor behavior is implementation-dependent; Table 36-5 summarizes several scenarios.

Table 36-5. Behavior on Restricted Memory Access

Scenario	Description
ToPA output region overlaps with SMRR, PRMRR, and SEAMRR protected regions	Stores to the restricted memory region will be dropped, and that packet data will be lost. Any attempt to read from that restricted region will return all 1s. The processor also may signal an error (Section 36.3.10) and disable tracing when the output pointer reaches the restricted region. If packet generation remains enabled, then packet output may continue once stores are no longer directed to restricted memory (on wrap, or if the output region is larger than the restricted memory region).
ToPA table overlaps with SMRR, PRMRR, and SEAMRR protected regions	The processor will signal an error (Section 36.3.10) and disable tracing when the ToPA write pointer (IA32_RTIT_OUTPUT_BASE + proc_trace_table_offset) enters the restricted region.

It should also be noted that packet output should not be routed to the 4KB APIC MMIO region, as defined by the IA32_APIC_BASE MSR. For details about the APIC, refer to the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A. No error is signaled for this case.

1. Usually, MAXPHYADDR is the value enumerated in CPUID.80000008H:EBX[7:0] (at most 52). If IA32_TME_ACTIVATE[0]= 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.

Modifications to Restricted Memory Regions

It is recommended that software disable packet generation before modifying the SMRRs to change the scope of the SMRR regions. This is because the processor reserves the right to cache any number of ToPA table entries internally, after checking them against restricted memory ranges. Once cached, the entries will not be checked again, meaning one could potentially route packet output to a newly restricted region. Software can ensure that any cached entries are written to memory by clearing IA32_RTIT_CTL.TraceEn.

36.2.8 Enabling and Configuration MSRs

36.2.8.1 General Considerations

Trace packet generation is enabled and configured by a collection of model-specific registers (MSRs), which are detailed below. Some notes on the configuration MSR behavior:

- If Intel Processor Trace is not supported by the processor (see Section 36.3.1), RDMSR or WRMSR of the IA32_RTIT_* MSRs will cause #GP.
- A WRMSR to any of the IA32_RTIT_* configuration MSRs while packet generation is enabled (IA32_RTIT_CTL.TraceEn=1) will generate a #GP exception. Packet generation must be disabled before the configuration MSRs can be changed.
Note: Software may write the same value back to IA32_RTIT_CTL without #GP, even if TraceEn=1.
- All configuration MSRs for Intel PT are duplicated per logical processor
- For each configuration MSR, any MSR write that attempts to change bits marked reserved, or utilize encodings marked reserved, will cause a #GP fault.
- All configuration MSRs for Intel PT are cleared on a warm or cold RESET.
 - If CPUID.14H.00H:EBX[2] = 1, only the TraceEn bit is cleared on warm RESET; though this may have the impact of clearing other bits in IA32_RTIT_STATUS. Other MSR values of the trace configuration MSRs are preserved on warm RESET.
- The semantics of MSR writes to trace configuration MSRs in this chapter generally apply to explicit WRMSR to these registers, using VMexit or VM entry MSR load list to these MSRs, XRSTORS with requested feature bit map including XSAVE map component of state_8 (corresponding to IA32_XSS[bit 8]), and the write to IA32_RTIT_CTL.TraceEn by XSAVES (Section 36.3.5.2).

36.2.8.2 IA32_RTIT_CTL MSR

IA32_RTIT_CTL, at address 570H, is the primary enable and control MSR for trace packet generation. Bit positions are listed in Table 36-6.

Table 36-6. IA32_RTIT_CTL MSR

Position	Bit Name	At Reset	Bit Description
0	TraceEn	0	If 1, enables tracing; else tracing is disabled. When this bit transitions from 1 to 0, all buffered packets are flushed out of internal buffers. A further store, fence, or architecturally serializing instruction may be required to ensure that packet data can be observed at the trace endpoint. See Section 36.2.8.3 for details of enabling and disabling packet generation. Note that the processor will clear this bit on #SMI (Section 36.2.9.3) and warm reset. Other MSR bits of IA32_RTIT_CTL (and other trace configuration MSRs) are not impacted by these events.
1	CYCEn	0	0: Disables CYC Packet (see Section 36.4.2.14). 1: Enables CYC Packet. This bit is reserved if CPUID.14H.00H:EBX[1] = 0.

Table 36-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
2	OS	0	0: Packet generation is disabled when CPL = 0. 1: Packet generation may be enabled when CPL = 0.
3	User	0	0: Packet generation is disabled when CPL > 0. 1: Packet generation may be enabled when CPL > 0.
4	PwrEvtEn	0	0: Power Event Trace packets are disabled. 1: Power Event Trace packets are enabled (see Section 36.2.3, “Power Event Tracing”).
5	FUPonPTW	0	0: PTW packets are not followed by FUPs. 1: PTW packets are followed by FUPs. This bit is reserved when CPUID.14H.00H:EBX[4] (“PTWRITE Supported”) is 0.
6	FabricEn	0	0: Trace output is directed to the memory subsystem, mechanism depends on IA32_RTIT_CTL.ToPA. 1: Trace output is directed to the trace transport subsystem, IA32_RTIT_CTL.ToPA is ignored. This bit is reserved if CPUID.14H.00H:ECX[3] = 0.
7	CR3Filter	0	0: Disables CR3 filtering. 1: Enables CR3 filtering. This bit is reserved if CPUID.14H.00H:EBX[0] (“CR3 Filtering Support”) is 0.
8	ToPA	0	0: Single-range output scheme enabled if CPUID.14H.00H:ECX.SNGL_RNG_OUT[2] = 1 and IA32_RTIT_CTL.FabricEn=0. 1: ToPA output scheme enabled (see Section 36.2.7.2) if CPUID.14H.00H:ECX.TOPAOUT[0] = 1, and IA32_RTIT_CTL.FabricEn=0. Note: WRMSR to IA32_RTIT_CTL that sets TraceEn but clears this bit and FabricEn would cause #GP, if CPUID.14H.00H:ECX.SNGL_RNG_OUT[2] = 0. WRMSR to IA32_RTIT_CTL that sets this bit causes #GP, if CPUID.14H.00H:ECX.TOPAOUT[0] = 0.
9	MTCEn	0	0: Disables MTC Packet (see Section 36.4.2.16). 1: Enables MTC Packet. This bit is reserved if CPUID.14H.00H:EBX[3] = 0.
10	TSCEn	0	0: Disable TSC packets. 1: Enable TSC packets (see Section 36.4.2.11).
11	DisRETC	0	0: Enable RET compression. 1: Disable RET compression (see Section 36.2.1.2).
12	PTWEn	0	0: PTWRITE packet generation disabled. 1: PTWRITE packet generation enabled (see Table 36-40 “PTW Packet Definition”). This bit is reserved when CPUID.14H.00H:EBX[4] (“PTWRITE Supported”) is 0.
13	BranchEn	0	0: Disable COFI-based packets. 1: Enable COFI-based packets: FUP, TIP, TIP.PGE, TIP.PGD, TNT, MODE.Exec, MODE.TSX. See Section 36.2.6.4 for details on BranchEn.

Table 36-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
17:14	MTCFreq	0	Defines MTC packet Frequency, which is based on the core crystal clock, or Always Running Timer (ART). MTC will be sent each time the selected ART bit toggles. The following Encodings are defined: 0: ART(0), 1: ART(1), 2: ART(2), 3: ART(3), 4: ART(4), 5: ART(5), 6: ART(6), 7: ART(7), 8: ART(8), 9: ART(9), 10: ART(10), 11: ART(11), 12: ART(12), 13: ART(13), 14: ART(14), 15: ART(15) Software must use CPUID to query the supported encodings in the processor, see Section 36.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.14H.00H:EBX[3] = 0.
18	Reserved	0	Must be 0.
22:19	CycThresh	0	CYC packet threshold, see Section 36.3.6 for details. CYC packets will be sent with the first eligible packet after N cycles have passed since the last CYC packet. If CycThresh is 0 then N=0, otherwise N is defined as $2^{(\text{CycThresh}-1)}$. The following Encodings are defined: 0: 0, 1: 1, 2: 2, 3: 4, 4: 8, 5: 16, 6: 32, 7: 64, 8: 128, 9: 256, 10: 512, 11: 1024, 12: 2048, 13: 4096, 14: 8192, 15: 16384 Software must use CPUID to query the supported encodings in the processor, see Section 36.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.14H.00H:EBX[1] = 0.
23	Reserved	0	Must be 0.
27:24	PSBFreq	0	Indicates the frequency of PSB packets. PSB packet frequency is based on the number of Intel PT packet bytes output, so this field allows the user to determine the increment of IA32_RTIT_STATUS.PacketByteCnt that should cause a PSB to be generated. Note that PSB insertion is not precise, but the average output bytes per PSB should approximate the SW selected period. The following Encodings are defined: 0: 2K, 1: 4K, 2: 8K, 3: 16K, 4: 32K, 5: 64K, 6: 128K, 7: 256K, 8: 512K, 9: 1M, 10: 2M, 11: 4M, 12: 8M, 13: 16M, 14: 32M, 15: 64M Software must use CPUID to query the supported encodings in the processor, see Section 36.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.14H.00H:EBX[1] = 0.
30:28	Reserved	0	Must be 0.
31	EventEn	0	0: Event Trace packets are disabled. 1: Event Trace packets are enabled. This bit is reserved when CPUID.14H.00H:EBX[7] ("Event Trace Supported") is 0.
35:32	ADDR0_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR0_A/B based on the following encodings: 0: ADDR0 range unused. 1: The [IA32_RTIT_ADDR0_A..IA32_RTIT_ADDR0_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 36.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR0_A..IA32_RTIT_ADDR0_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See 4.2.8 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.14H.01H:EAX.RANGECONT[2:0] < 1.

Table 36-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
39:36	ADDR1_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR1_A/B based on the following encodings: 0: ADDR1 range unused. 1: The [IA32_RTIT_ADDR1_A..IA32_RTIT_ADDR1_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 36.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR1_A..IA32_RTIT_ADDR1_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 36.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.14H.01H:EAX.RANGECONT[2:0] < 2.
43:40	ADDR2_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR2_A/B based on the following encodings: 0: ADDR2 range unused. 1: The [IA32_RTIT_ADDR2_A..IA32_RTIT_ADDR2_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 36.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR2_A..IA32_RTIT_ADDR2_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 36.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.14H.01H:EAX.RANGECONT[2:0] < 3.
47:44	ADDR3_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR3_A/B based on the following encodings: 0: ADDR3 range unused. 1: The [IA32_RTIT_ADDR3_A..IA32_RTIT_ADDR3_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 36.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR3_A..IA32_RTIT_ADDR3_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 36.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.14H.01H:EAX.RANGECONT[2:0] < 4.
54:48	Reserved	0	Reserved only for future trace content enables, or address filtering configuration enables. Must be 0.
55	DisTNT	0	0: Include TNT packets in control flow trace. 1: Omit TNT packets from control flow trace. This bit is reserved when CPUID.14H.00H:EBX[8] ("TNT Disable Supported") is 0. See Section 36.3.9 for details.
56	InjectPsbPmiOnEnable	0	1: Enables use of IA32_RTIT_STATUS bits PendPSB[6] and PendTopaPMI[7], see Section 36.2.8.4, "IA32_RTIT_STATUS MSR," for behavior of these bits. 0: IA32_RTIT_STATUS bits 6 and 7 are ignored. This field is reserved if CPUID.14H.00H:EBX[6] = 0.
59:57	Reserved	0	Reserved only for future trace content enables, or address filtering configuration enables. Must be 0.
63:60	Reserved	0	Must be 0.

36.2.8.3 Enabling and Disabling Packet Generation with TraceEn

When TraceEn transitions from 0 to 1, Intel Processor Trace is enabled, and a series of packets may be generated. These packets help ensure that the decoder is aware of the state of the processor when the trace begins, and that it can keep track of any timing or state changes that may have occurred while packet generation was disabled. A full PSB+ (see Section 36.4.2.17) will be generated if IA32_RTIT_STATUS.PacketByteCnt=0, and may be generated in other cases as well. Otherwise, timing packets will be generated, including TSC, TMA, and CBR (see Section 36.4.1.1).

In addition to the packets discussed above, if and when PacketEn (Section 36.2.6.1) transitions from 0 to 1 (which may happen immediately, depending on filtering settings), a TIP.PGE packet (Section 36.4.2.3) will be generated.

When TraceEn is set, the processor may read ToPA entries from memory and cache them internally. For this reason, software should disable packet generation before making modifications to the ToPA tables (or changing the configuration of restricted memory regions). See Section 36.7 for more details of packets that may be generated with modifications to TraceEn.

Disabling Packet Generation

Clearing TraceEn causes any packet data buffered within the logical processor to be flushed out, after which the output MSRs (IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS) will have stable values. When output is directed to memory, a store, fence, or architecturally serializing instruction may be required to ensure that the packet data is globally observed. No special packets are generated by disabling packet generation, though a TIP.PGD may result if PacketEn=1 at the time of disable.

Other Writes to IA32_RTIT_CTL

Any attempt to modify IA32_RTIT_CTL while TraceEn is set will result in a general-protection fault (#GP) unless the same write also clears TraceEn. However, writes to IA32_RTIT_CTL that do not modify any bits will not cause a #GP, even if TraceEn remains set.

36.2.8.4 IA32_RTIT_STATUS MSR

The IA32_RTIT_STATUS MSR is readable and writable by software, though some fields cannot be modified by software. See Table 36-7 for details. The WRMSR instruction ignores these bits in the source operand (attempts to modify these bits are ignored and do not cause WRMSR to fault).

This MSR can only be written when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP). The processor does not modify the value of this MSR while TraceEn is 0 (software can modify it with WRMSR).

Table 36-7. IA32_RTIT_STATUS MSR

Position	Bit Name	At Reset	Bit Description
0	FilterEn	0	This bit is written by the processor, and indicates that tracing is allowed for the current IP, see Section 36.2.6.5. Writes are ignored.
1	ContextEn	0	The processor sets this bit to indicate that tracing is allowed for the current context. See Section 36.2.6.3. Writes are ignored.
2	TriggerEn	0	The processor sets this bit to indicate that tracing is enabled. See Section 36.2.6.2. Writes are ignored.
3	Reserved	0	Must be 0.
4	Error	0	The processor sets this bit to indicate that an operational error has been encountered. When this bit is set, TriggerEn is cleared to 0 and packet generation is disabled. For details, see “ToPA Errors” in Section 36.2.7.2. When TraceEn is cleared, software can write this bit. Once it is set, only software can clear it. It is not recommended that software ever set this bit, except in cases where it is restoring a prior saved state.

Table 36-7. IA32_RTIT_STATUS MSR

Position	Bit Name	At Reset	Bit Description
5	Stopped	0	The processor sets this bit to indicate that a ToPA Stop condition has been encountered. When this bit is set, TriggerEn is cleared to 0 and packet generation is disabled. For details, see “ToPA STOP” in Section 36.2.7.2. When TraceEn is cleared, software can write this bit. Once it is set, only software can clear it. It is not recommended that software ever set this bit, except in cases where it is restoring a prior saved state.
6	PendPSB	0	If IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1, the processor sets this bit when the threshold for a PSB+ to be inserted has been reached. The processor will clear this bit when the PSB+ has been inserted into the trace. If PendPSB = 1 and InjectPsbPmiOnEnable = 1 when IA32_RTIT_CTL.TraceEn[0] transitions from 0 to 1, a PSB+ will be inserted into the trace. This field is reserved if CPUID.14H.00H:EBX[6] = 0.
7	PendTopaPMI	0	If IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1, the processor sets this bit when the threshold for a ToPA PMI to be inserted has been reached. Software should clear this bit once the ToPA PMI has been handled, see “ToPA PMI” for details. If PendTopaPMI = 1 and InjectPsbPmiOnEnable = 1 when IA32_RTIT_CTL.TraceEn[0] transitions from 0 to 1, a PMI will be pended. This field is reserved if CPUID.14H.00H:EBX[6] = 0.
31:8	Reserved	0	Must be 0.
48:32	PacketByteCnt	0	This field is written by the processor, and holds a count of packet bytes that have been sent out. The processor also uses this field to determine when the next PSB packet should be inserted. Note that the processor may clear or modify this field at any time while IA32_RTIT_CTL.TraceEn=1. It will have a stable value when IA32_RTIT_CTL.TraceEn=0. See Section 36.4.2.17 for details. This field is reserved when CPUID.14H.00H:EBX[1] (“Configurable PSB and CycleAccurate Mode Supported”) is 0.
63:49	Reserved	0	Must be 0.

36.2.8.5 IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B MSRs

The role of the IA32_RTIT_ADDRn_A/B register pairs, for each n, is determined by the corresponding ADDRn_CFG fields in IA32_RTIT_CTL (see Section 36.2.8.2). The number of these register pairs is enumerated by CPUID.14H.01H:EAX.RANGEcnt[2:0].

- Processors that enumerate support for 1 range support:
 - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
- Processors that enumerate support for 2 ranges support:
 - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
 - IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B
- Processors that enumerate support for 3 ranges support:
 - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
 - IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B
 - IA32_RTIT_ADDR2_A, IA32_RTIT_ADDR2_B
- Processors that enumerate support for 4 ranges support:
 - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
 - IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B
 - IA32_RTIT_ADDR2_A, IA32_RTIT_ADDR2_B

— IA32_RTIT_ADDR3_A, IA32_RTIT_ADDR3_B

Each register has a single 64-bit field that holds a linear address value. Writes must ensure that the address is in canonical form, otherwise a general-protection fault (#GP) fault will result.

Each MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP).

36.2.8.6 IA32_RTIT_CR3_MATCH MSR

The IA32_RTIT_CR3_MATCH register is compared against CR3 when IA32_RTIT_CTL.CR3Filter is 1. Bits 63:5 hold the CR3 address value to match, bits 4:0 are reserved to 0. For more details on CR3 filtering and the treatment of this register, see Section 36.2.5.2.

This MSR is accessible if CPUID.14H.00H:EBX[0], “CR3 Filtering Support”, is 1. This MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP). IA32_RTIT_CR3_MATCH[4:0] are reserved and must be 0; an attempt to set those bits using WRMSR causes a #GP.

36.2.8.7 IA32_RTIT_OUTPUT_BASE MSR

This MSR is used to configure the trace output destination, when output is directed to memory (IA32_RTIT_CTL.FabricEn = 0). The size of the address field is determined by two values related to the processor’s physical-address width, MAXPHYADDR:

- M is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52).
- MAXPHYADDR is typically the same as M. However, if IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.

See Table 36-8 for details.

WRMSR and XRSTORS will fault if any of bits 63:M would be set to 1; they allow setting bits M–1:MAXPHYADDR; however, if any of those bits are set outside SEAM, trace output may fail in a model-specific manner.

When the ToPA output scheme is used, the processor may update this MSR when packet generation is enabled, and those updates are asynchronous to instruction execution. Therefore, the values in this MSR should be considered unreliable unless packet generation is disabled (IA32_RTIT_CTL.TraceEn = 0).

Accesses to this MSR are supported only if Intel PT output to memory is supported, hence when either CPUID.14H.00H:ECX[0] or CPUID.14H.00H:ECX[2] are set. Otherwise WRMSR or RDMSR cause a general-protection fault (#GP). If supported, this MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP).

Table 36-8. IA32_RTIT_OUTPUT_BASE MSR

Position	Bit Name	At Reset	Bit Description
6:0	Reserved	0	Must be 0.
MAXPHYADDR-1:7	BasePhysAddr	0	The base physical address. How this address is used depends on the value of IA32_RTIT_CTL.ToPA: 0: This is the base physical address of a single, contiguous physical output region. This could be mapped to DRAM or to MMIO, depending on the value. The base address should be aligned with the size of the region, such that none of the 1s in the mask value (Section 36.2.8.8) overlap with 1s in the base address. If the base is not aligned, an operational error will result (see Section 36.3.10). 1: The base physical address of the current ToPA table. The address must be 4K aligned. Writing an address in which bits 11:7 are non-zero will not cause a #GP, but an operational error will be signaled once TraceEn is set. See “ToPA Errors” in Section 36.2.7.2, as well as Section 36.3.10.
M-1:MAXPHYADDR	Reserved	0	Should be 0 when processor trace is enabled.
63:M	Reserved	0	Must be 0.

36.2.8.8 IA32_RTIT_OUTPUT_MASK_PTRS MSR

This MSR holds any mask or pointer values needed to indicate where the next byte of trace output should be written. The meaning of the values held in this MSR depend on whether the ToPA output mechanism is in use. See Section 36.2.7.2 for details.

The processor updates this MSR while when packet generation is enabled, and those updates are asynchronous to instruction execution. Therefore, the values in this MSR should be considered unreliable unless packet generation is disabled (IA32_RTIT_CTL.TraceEn = 0).

Accesses to this MSR are supported only if Intel PT output to memory is supported, hence when either CPUID.14H.00H:ECX[0] or CPUID.14H.00H:ECX[2] are set. Otherwise WRMSR or RDMSR cause a general-protection fault (#GP). If supported, this MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP).

Table 36-9. IA32_RTIT_OUTPUT_MASK_PTRS MSR

Position	Bit Name	At Reset	Bit Description
6:0	LowerMask	7FH	Forced to 1, writes are ignored.
31:7	MaskOrTableOffset	0	The use of this field depends on the value of IA32_RTIT_CTL.ToPA: 0: This field holds bits 31:7 of the mask value for the single, contiguous physical output region. The size of this field indicates that regions can be of size 128B up to 4GB. This value (combined with the lower 7 bits, which are reserved to 1) will be ANDed with the OutputOffset field to determine the next write address. All 1s in this field should be consecutive and starting at bit 7, otherwise the region will not be contiguous, and an operational error (Section 36.3.10) will be signaled when TraceEn is set. 1: This field holds bits 27:3 of the offset pointer into the current ToPA table. This value can be added to the IA32_RTIT_OUTPUT_BASE value to produce a pointer to the current ToPA table entry, which itself is a pointer to the current output region. In this scenario, the lower 7 reserved bits are ignored. This field supports tables up to 256 MBytes in size.
63:32	OutputOffset	0	The use of this field depends on the value of IA32_RTIT_CTL.ToPA: 0: This is bits 31:0 of the offset pointer into the single, contiguous physical output region. This value will be added to the IA32_RTIT_OUTPUT_BASE value to form the physical address at which the next byte of packet output data will be written. This value must be less than or equal to the MaskOrTableOffset field, otherwise an operational error (Section 36.3.10) will be signaled when TraceEn is set. 1: This field holds bits 31:0 of the offset pointer into the current ToPA output region. This value will be added to the output region base field, found in the current ToPA table entry, to form the physical address at which the next byte of trace output data will be written. This value must be less than the ToPA entry size, otherwise an operational error (Section 36.3.10) will be signaled when TraceEn is set.

36.2.9 Interaction of Intel® Processor Trace and Other Processor Features

36.2.9.1 Intel® Transactional Synchronization Extensions (Intel® TSX)

The operation of Intel TSX is described in Chapter 14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1. For tracing purpose, packet generation does not distinguish between hardware lock elision (HLE) and restricted transactional memory (RTM), but speculative execution does have impacts on the trace output. Specifically, packets are generated as instructions complete, even for instructions in a transactional region that is later aborted. For this reason, debugging software will need indication of the beginning and end of a transactional region; this will allow software to understand when instructions are part of a transactional region and whether that region has been committed.

To enable this, TSX information is included in a MODE packet leaf. The mode bits in the leaf are:

- **InTX:** Set to 1 on an TSX transaction begin, and cleared on transaction commit or abort.

- **TXAbort:** Set to 1 only when InTX transitions from 1 to 0 on an abort. Cleared otherwise.

If BranchEn=1, this MODE packet will be sent each time the transaction status changes. See Table 36-10 for details.

Table 36-10. TSX Packet Scenarios with BranchEn=1

TSX Event	Instruction	Packets
Transaction Begin	Either XBEGIN or XACQUIRE lock (the latter if executed transactionally)	MODE(TXAbort=0, InTX=1), FUP(CurrentIP)
Transaction Commit	Either XEND or XRELEASE lock, if transactional execution ends. This happens only on the outermost commit	MODE(TXAbort=0, InTX=0), FUP(CurrentIP)
Transaction Abort	XABORT or other transactional abort	MODE(TXAbort=1, InTX=0), FUP(CurrentIP), TIP(TargetIP)
Other	One of the following: ▪ Nested XBEGIN or XACQUIRE lock ▪ An outer XACQUIRE lock that doesn't begin a transaction (InTX not set) ▪ Non-outermost XEND or XRELEASE lock	None. No change to TSX mode bits for these cases.

The CurrentIP listed above is the IP of the associated instruction. The TargetIP is the IP of the next instruction to be executed; for HLE, this is the XACQUIRE lock; for RTM, this is the fallback handler.

Intel PT stores are non-transactional, and thus packet writes are not rolled back on TSX abort.

36.2.9.2 TSX and IP Filtering

A complication with tracking transactions is handling transactions that start or end outside of the tracing region. Transactions can't span across a change in ContextEn, because CPL changes and CR3 changes each cause aborts. But a transaction can start within the IP filter region and end outside it.

To assist the decoder handling this situation, MODE.TSX packets can be sent even if FilterEn=0, though there will be no FUP attached. Instead, they will merely serve to indicate to the decoder when transactions are active and when they are not. When tracing resumes (due to PacketEn=1), the last MODE.TSX preceding the TIP.PGE will indicate the current transaction status.

36.2.9.3 System Management Mode (SMM)

SMM code has special privileges that non-SMM code does not have. Intel Processor Trace can be used to trace SMM code, but special care is taken to ensure that SMM handler context is not exposed in any non-SMM trace collection. Additionally, packet output from tracing non-SMM code cannot be written into memory space that is either protected by SMRR or used by the SMM handler.

SMM is entered via a system management interrupt (SMI). SMI delivery saves the value of IA32_RTIT_CTL.TraceEn into SMRAM and then clears it, thereby disabling packet generation.

The saving and clearing of IA32_RTIT_CTL.TraceEn ensures two things:

1. All internally buffered packet data is flushed before entering SMM (see Section 36.2.8.2).
2. Packet generation ceases before entering SMM, so any tracing that was configured outside SMM does not continue into SMM. No SMM instruction pointers or other state will be exposed in the non-SMM trace.

When the RSM instruction is executed to return from SMM, the TraceEn value that was saved by SMI delivery is restored, allowing tracing to be resumed. As is done any time packet generation is enabled, ContextEn is re-evaluated, based on the values of CPL, CR3, etc., established by RSM.

Like other interrupts, delivery of an SMI produces a FUP containing the IP of the next instruction to execute. By toggling TraceEn, SMI and RSM can produce TIP.PGD and TIP.PGE packets, respectively, indicating that tracing was disabled or re-enabled. See Table 36.7 for more information about packets entering and leaving SMM.

Although #SMI and RSM change CR3, PIP packets are not generated in these cases. With #SMI tracing is disabled before the CR3 change; with RSM TraceEn is restored after CR3 is written.

TraceEn must be cleared before executing RSM, otherwise it will cause a shutdown. Further, on processors that restrict use of Intel PT with LBRs (see Section 36.3.1.2), any RSM that results in enabling of both will cause a shutdown.

Intel PT can support tracing of System Transfer Monitor operating in SMM, see Section 36.6.

36.2.9.4 Virtual-Machine Extensions (VMX)

Initial implementations of Intel Processor Trace do not support tracing in VMX operation. Such processors indicate this by returning 0 for IA32_VMX_MISC[bit 14]. On these processors, execution of the VMXON instruction clears IA32_RTIT_CTL.TraceEn and any attempt to write IA32_RTIT_CTL in VMX operation causes a general-protection exception (#GP).

Processors that support Intel Processor Trace in VMX operation return 1 for IA32_VMX_MISC[bit 14]. Details of tracing in VMX operation are described in Section 36.4.2.26.

36.2.9.5 Intel® Software Guard Extensions (Intel® SGX)

Intel SGX provides an application with the ability to instantiate a protective container (an enclave) with confidentiality and integrity (see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D). On a processor with both Intel PT and Intel SGX enabled, when executing code within a production enclave, no control flow packets are produced by Intel PT. An enclave entry will clear ContextEn, thereby blocking control flow packet generation. A TIP.PGD packet will be generated if PacketEn=1 at the time of the entry.

Upon enclave exit, ContextEn will no longer be forced to 0. If other enables are set at the time, a TIP.PGE may be generated to indicate that tracing is resumed.

During the enclave execution, Intel PT remains enabled, and periodic or timing packets such as PSB, TSC, MTC, or CBR can still be generated. No IPs or other architectural state will be exposed.

For packet generation examples on enclave entry or exit, see Section 36.7.

Debug Enclaves

Intel SGX allows an enclave to be configured with relaxed protection of confidentiality for debug purposes, see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D. In a debug enclave, Intel PT continues to function normally. Specifically, ContextEn is not impacted by an enclave entry or exit. Hence, the generation of ContextEn-dependent packets within a debug enclave is allowed.

36.2.9.6 SENTER/ENTERACCS and ACM

GETSEC[SENDER] and GETSEC[ENTERACCS] instructions clear TraceEn, and it is not restored when those instructions complete. SENTER also causes TraceEn to be cleared on other logical processors when they rendezvous and enter the SENTER sleep state. In these two cases, the disabling of packet generation is not guaranteed to flush internally buffered packets. Some packets may be dropped.

When executing an authenticated code module (ACM), packet generation is silently disabled during ACRAM setup. TraceEn will be cleared, but no TIP.PGD packet is generated. After completion of the module, the TraceEn value will be restored. There will be no TIP.PGE packet, but timing packets, like TSC and CBR, may be produced.

36.2.9.7 Intel® Memory Protection Extensions (Intel® MPX)

Bounds exceptions (#BR) caused by Intel MPX are treated like other exceptions, producing FUP and TIP packets that indicate the source and destination IPs.

36.3 CONFIGURATION AND PROGRAMMING GUIDELINE

36.3.1 Detection of Intel Processor Trace and Capability Enumeration

Processor support for Intel Processor Trace is indicated by CPUID.07H.00H:EBX[25]= 1. CPUID function 14H is dedicated to enumerate the resource and capability of processors that report CPUID.07H.00H:EBX[25]= 1. Different processor generations may have architecturally-defined variation in capabilities. Table 36-11 describes details of the enumerable capabilities that software must use across generations of processors that support Intel Processor Trace.

Table 36-11. CPUID.14H Enumeration of Intel Processor Trace Capabilities

CPUID.14H.00H		Name	Description Behavior
Register	Bits		
EAX	31:0	Maximum valid sub-leaf Index	Specifies the index of the maximum valid sub-leaf for this CPUID leaf.
EBX	0	CR3 Filtering Support	1: Indicates that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed. See Section 36.2.8. 0: Indicates that writes that set IA32_RTIT_CTL.CR3Filter to 1, or any access to IA32_RTIT_CR3_MATCH, will generate a #GP exception.
	1	Configurable PSB and Cycle-Accurate Mode Supported	1: (a) IA32_RTIT_CTL.PSBFreq can be set to a non-zero value, in order to select the preferred PSB frequency (see below for allowed values). (b) IA32_RTIT_STATUS.PacketByteCnt can be set to a non-zero value, and will be incremented by the processor when tracing to indicate progress towards the next PSB. If trace packet generation is enabled by setting TraceEn, a PSB will only be generated if PacketByteCnt=0. (c) IA32_RTIT_CTL.CYCEn can be set to 1 to enable Cycle-Accurate Mode. See Section 36.2.8. 0: (a) Any attempt to write a non-zero value to IA32_RTIT_CTL.PSBFreq or IA32_RTIT_STATUS.PacketByteCnt will generate a #GP exception. (b) If trace packet generation is enabled by setting TraceEn, a PSB is always generated. (c) Any attempt to write a non-zero value to IA32_RTIT_CTL.CYCEn or IA32_RTIT_CTL.CycThresh will generate a #GP exception.
	2	IP Filtering and TraceStop supported, and Preserve Intel PT MSRs across warm reset	1: (a) IA32_RTIT_CTL provides at one or more ADDRn_CFG field to configure the corresponding address range MSRs for IP Filtering or IP TraceStop. Each ADDRn_CFG field accepts a value in the range of 0:2 inclusive. The number of ADDRn_CFG fields is reported by CPUID.14H.01H:EAX.RANGEcnt[2:0]. (b) At least one register pair IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B are provided to configure address ranges for IP filtering or IP TraceStop. (c) On warm reset, all Intel PT MSRs will retain their pre-reset values, though IA32_RTIT_CTL.TraceEn will be cleared. The Intel PT MSRs are listed in Section 36.2.8. 0: (a) An Attempt to write IA32_RTIT_CTL.ADDRn_CFG with non-zero encoding values will cause #GP. (b) Any access to IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B, will generate a #GP exception. (c) On warm reset, all Intel PT MSRs will be cleared.
	3	MTC Supported	1: IA32_RTIT_CTL.MTCEn can be set to 1, and MTC packets will be generated. See Section 36.2.8. 0: An attempt to set IA32_RTIT_CTL.MTCEn or IA32_RTIT_CTL.MTCFreq to a non-zero value will generate a #GP exception.

Table 36-11. CPUID.14H Enumeration of Intel Processor Trace Capabilities (Contd.)

CPUID.14H.00H		Name	Description Behavior
Register	Bits		
	4	PTWRITE Supported	1: Writes can set IA32_RTIT_CTL[12] (PTWEn) and IA32_RTIT_CTL[5] (FUPonPTW), and PTWRITE can generate packets. 0: Writes that set IA32_RTIT_CTL[12] or IA32_RTIT_CTL[5] will generate a #GP exception, and PTWRITE will #UD fault.
	5	Power Event Trace Supported	1: Writes can set IA32_RTIT_CTL[4] (PwrEvtEn), enabling Power Event Trace packet generation. 0: Writes that set IA32_RTIT_CTL[4] will generate a #GP exception.
	6	PSB and PMI Preservation Supported	1: Writes can set IA32_RTIT_CTL[56] (InjectPsbPmiOnEnable), enabling the processor to set IA32_RTIT_STATUS[7] (PendTopaPMI) and/or IA32_RTIT_STATUS[6] (PendPSB) in order to preserve ToPA PMIs and/or PSBs otherwise lost due to Intel PT disable. Writes can also set PendToPAPMI and PendPSB. 0: Writes that set IA32_RTIT_CTL[56], IA32_RTIT_STATUS[7], or IA32_RTIT_STATUS[6] will generate a #GP exception.
	7	Event Trace Supported	1: Writes can set IA32_RTIT_CTL[31] (EventEn), enabling Event Trace packet generation. 0: Writes that set IA32_RTIT_CTL[31] will generate a #GP exception.
	8	TNT Disable Supported	1: Writes can set IA32_RTIT_CTL[55] (DisTNT), disabling TNT packet generation. 0: Writes that set IA32_RTIT_CTL[55] will generate a #GP exception.
	31:9	Reserved	
ECX	0	ToPA Output Supported	1: Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme (Section 36.2.7.2) IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed. 0: Unless CPUID.14H.00H:ECX.SNGL_RNG_OUT[2] = 1, writes to IA32_RTIT_OUTPUT_BASE or IA32_RTIT_OUTPUT_MASK_PTRS MSRs will generate a #GP exception.
	1	ToPA Tables Allow Multiple Output Entries	1: ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOrTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS. 0: ToPA tables can hold only one output entry, which must be followed by an END=1 entry which points back to the base of the table. Further, ToPA PMIs will be delivered before the region is filled. See ToPA PMI in Section 36.2.7.2. If there is more than one output entry before the END entry, or if the END entry has the wrong base address, an operational error will be signaled (see “ToPA Errors” in Section 36.2.7.2).
	2	Single-Range Output Supported	1: Enabling tracing (TraceEn=1) with IA32_RTIT_CTL.ToPA=0 is supported. 0: Unless CPUID.14H.00H:ECX.TOPAOUT[0] = 1, writes to IA32_RTIT_OUTPUT_BASE or IA32_RTIT_OUTPUT_MASK_PTRS MSRs will generate a #GP exception.
	3	Output to Trace Transport Subsystem Supported	1: Setting IA32_RTIT_CTL.FabricEn to 1 is supported. 0: IA32_RTIT_CTL.FabricEn is reserved. Write 1 to IA32_RTIT_CTL.FabricEn will generate a #GP exception.
	30:4	Reserved	

Table 36-11. CPUID.14H Enumeration of Intel Processor Trace Capabilities (Contd.)

CPUID.14H.00H		Name	Description Behavior
Register	Bits		
	31	IP Payloads are LIP	1: Generated packets which contain IP payloads have LIP values, which include the CS base component. 0: Generated packets which contain IP payloads have RIP values, which are the offset from CS base.
EDX	31:0	Reserved	

If CPUID.14H.00H:EAX reports a non-zero value, additional capabilities of Intel Processor Trace are described in the sub-leaves of CPUID.14H.

Table 36-12. CPUID.14H.01H Enumeration of Intel Processor Trace Capabilities

CPUID.14H.01H		Name	Description Behavior
Register	Bits		
EAX	2:0	Number of Address Ranges	A non-zero value specifies the number ADDRn_CFG field supported in IA32_RTIT_CTL and the number of register pair IA32_RTIT_ADDRn_A/IA32_RTIT_ADDRn_B supported for IP filtering and IP TraceStop. NOTE: Currently, no processors support more than 4 address ranges.
	15:3	Reserved	
	31:16	Bitmap of supported MTC Period Encodings	The non-zero bits indicate the map of supported encoding values for the IA32_RTIT_CTL.MTCFreq field. This applies only if CPUID.14H.00H:EBX[3] = 1 (MTC Packet generation is supported), otherwise the MTCFreq field is reserved to 0. Each bit position in this field represents 1 encoding value in the 4-bit MTCFreq field (ie, bit 0 is associated with encoding value 0). For each bit: 1: MTCFreq can be assigned the associated encoding value. 0: MTCFreq cannot be assigned to the associated encoding value. A write to IA32_RTIT_CTL.MTCFreq with unsupported encoding will cause #GP fault.
EBX	15:0	Bitmap of supported Cycle Threshold values	The non-zero bits indicate the map of supported encoding values for the IA32_RTIT_CTL.CycThresh field. This applies only if CPUID.14H.00H:EBX[1] = 1 (Cycle-Accurate Mode is Supported), otherwise the CycThresh field is reserved to 0. See Section 36.2.8. Each bit position in this field represents 1 encoding value in the 4-bit CycThresh field (ie, bit 0 is associated with encoding value 0). For each bit: 1: CycThresh can be assigned the associated encoding value. 0: CycThresh cannot be assigned to the associated encoding value. A write to CycThresh with unsupported encoding will cause #GP fault.

Table 36-12. CPUID.14H.01H Enumeration of Intel Processor Trace Capabilities (Contd.)

CPUID.14H.01H		Name	Description Behavior
Register	Bits		
	31:16	Bitmap of supported Configurable PSB Frequency encoding	The non-zero bits indicate the map of supported encoding values for the IA32_RTIT_CTL.PSBFreq field. This applies only if CPUID.14H.00H:EBX[1] = 1 (Configurable PSB is supported), otherwise the PSBFreq field is reserved to 0. See Section 36.2.8. Each bit position in this field represents 1 encoding value in the 4-bit PSBFreq field (ie, bit 0 is associated with encoding value 0). For each bit: 1: PSBFreq can be assigned the associated encoding value. 0: PSBFreq cannot be assigned to the associated encoding value. A write to PSBFreq with unsupported encoding will cause #GP fault.
ECX	31:0	Reserved	
EDX	31:0	Reserved	

36.3.1.1 Packet Decoding of RIP versus LIP

FUP, TIP, TIP.PGE, and TIP.PGE packets can contain an instruction pointer (IP) payload. On some processor generations, this payload will be an effective address (RIP), while on others this will be a linear address (LIP). In the former case, the payload is the offset from the current CS base address, while in the latter it is the sum of the offset and the CS base address (Note that in real mode, the CS base address is the value of CS<<4, while in protected mode the CS base address is the base linear address of the segment indicated by the CS register.). Which IP type is in use is indicated by enumeration (see CPUID.14H.00H:ECX.LIP[31] in Table 36-11).

For software that executes while the CS base address is 0 (including all software executing in 64-bit mode), the difference is indistinguishable. A trace decoder must account for cases where the CS base address is not 0 and the resolved LIP will not be evident in a trace generated on a CPU that enumerates use of RIP. This is likely to cause problems when attempting to link the trace with the associated binaries.

Note that IP comparison logic, for IP filtering and TraceStop range calculation, is based on the same IP type as these IP packets. For processors that output RIP, the IP comparison mechanism is also based on RIP, and hence on those processors RIP values should be written to IA32_RTIT_ADDRn_[AB] MSRs. This can produce differing behavior if the same trace configuration setting is run on processors reporting different IP types, i.e., CPUID.14H.00H:ECX.LIP[31]. Care should be taken to check CPUID when configuring IP filters.

36.3.1.2 Model Specific Capability Restrictions

Some processor generations impose restrictions that prevent use of LBRs/BTS/BTM/LEs when software has enabled tracing with Intel Processor Trace. On these processors, when TraceEn is set, updates of LBR, BTS, BTM, LEs are suspended but the states of the corresponding IA32_DEBUGCTL control fields remained unchanged as if it were still enabled. When TraceEn is cleared, the LBR array is reset, and LBR/BTS/BTM/LEs updates will resume. Further, reads of these registers will return 0, and writes will be dropped.

The list of MSRs whose updates/accesses are restricted follows.

- MSR_LASTBRANCH_x_TO_IP, MSR_LASTBRANCH_x_FROM_IP, MSR_LBR_INFO_x, MSR_LASTBRANCH_TOS
- MSR_LER_FROM_LIP, MSR_LER_TO_LIP
- MSR_LBR_SELECT

For processors with CPUID DisplayFamily_DisplayModel signatures of 06_3DH, 06_47H, 06_4EH, 06_4FH, 06_56H, and 06_5EH, the use of Intel PT and LBRs are mutually exclusive.

36.3.2 Enabling and Configuration of Trace Packet Generation

To configure trace packets, enable packet generation, and capture packets, software starts with using CPUID instruction to detect its feature flag, CPUID.07H.00H:EBX[25]= 1; followed by enumerating the capabilities described in Section 36.3.1.

Based on the capability queried from Section 36.3.1, software must configure a number of model-specific registers. This section describes programming considerations related to those MSRs.

36.3.2.1 Enabling Packet Generation

When configuring and enabling packet generation, the IA32_RTIT_CTL MSR should be written after any other Intel PT MSRs have been written, since writes to the other configuration MSRs cause a general-protection fault (#GP) if TraceEn = 1. If a prior trace collection context is not being restored, then software should first clear IA32_RTIT_STATUS. This is important since the Stopped, and Error fields are writable; clearing the MSR clears any values that may have persisted from prior trace packet collection contexts. See Section 36.2.8.2 for details of packets generated by setting TraceEn to 1.

If setting TraceEn to 1 causes an operational error (see Section 36.3.10), there may be a delay after the WRMSR completes before the error is signaled in the IA32_RTIT_STATUS MSR.

While packet generation is enabled, the values of some configuration MSRs (e.g., IA32_RTIT_STATUS and IA32_RTIT_OUTPUT_*) are transient, and reads may return values that are out of date. Only after packet generation is disabled (by clearing TraceEn) do reads of these MSRs return reliable values.

36.3.2.2 Disabling Packet Generation

After disabling packet generation by clearing IA32_RTIT_CTL, it is advisable to read the IA32_RTIT_STATUS MSR (Section 36.2.8.4):

- If the Error bit is set, an operational error was encountered, and the trace is most likely compromised. Software should check the source of the error (by examining the output MSR values), correct the source of the problem, and then attempt to gather the trace again. For details on operational errors, see Section 36.3.10. Software should clear IA32_RTIT_STATUS.Error before re-enabling packet generation.
- If the Stopped bit is set, software execution encountered an IP TraceStop (see Section 36.2.5.3) or the ToPA Stop condition (see “ToPA STOP” in Section 36.2.7.2) before packet generation was disabled.

36.3.3 Flushing Trace Output

Packets are first buffered internally and then written out asynchronously. To collect packet output for post-processing, a collector needs first to ensure that all packet data has been flushed from internal buffers. Software can ensure this by stopping packet generation by clearing IA32_RTIT_CTL.TraceEn (see “Disabling Packet Generation” in Section 36.2.8.2).

When software clears IA32_RTIT_CTL.TraceEn to flush out internally buffered packets, the logical processor issues an SFENCE operation which ensures that WC trace output stores will be ordered with respect to the next store, or serializing operation. A subsequent read from the same logical processor will see the flushed trace data, while a read from another logical processor should be preceded by a store, fence, or architecturally serializing operation on the tracing logical processor.

When the flush operations complete, the IA32_RTIT_OUTPUT_* MSR values indicate where the trace ended. While TraceEn is set, these MSRs may hold stale values. Further, if a ToPA region with INT=1 is filled, meaning a ToPA PMI has been triggered, IA32_PERF_GLOBAL_STATUS.Trace_ToPA_PMI[55] will be set by the time the flush completes.

36.3.4 Warm Reset

The MSRs software uses to program Intel Processor Trace are cleared after a power-on RESET (or cold RESET). On a warm RESET, the contents of those MSRs can retain their values from before the warm RESET with the exception that IA32_RTIT_CTL.TraceEn will be cleared (which may have the side effect of clearing some bits in IA32_RTIT_STATUS).

36.3.5 Context Switch Consideration

To facilitate construction of instruction execution traces at the granularity of a software process or thread context, software can save and restore the states of the trace configuration MSRs across the process or thread context switch boundary. The principle is the same as saving and restoring the typical architectural processor states across context switches.

36.3.5.1 Manual Trace Configuration Context Switch

The configuration can be saved and restored through a sequence of instructions of RDMSR, management of MSR content and WRMSR. To stop tracing and to ensure that all configuration MSRs contain stable values, software must clear IA32_RTIT_CTL.TraceEn before reading any other trace configuration MSRs. The recommended method for saving trace configuration context manually follows:

1. RDMSR IA32_RTIT_CTL, save value to memory
2. WRMSR IA32_RTIT_CTL with saved value from RDMSR above and TraceEn cleared
3. RDMSR all other configuration MSRs whose values had changed from previous saved value, save changed values to memory

When restoring the trace configuration context, IA32_RTIT_CTL should be restored last:

1. Read saved configuration MSR values, aside from IA32_RTIT_CTL, from memory, and restore them with WRMSR
2. Read saved IA32_RTIT_CTL value from memory, and restore with WRMSR.

36.3.5.2 Trace Configuration Context Switch Using XSAVES/XRSTORS

On processors whose XSAVE feature set supports XSAVES and XRSTORS, the Trace configuration state can be saved using XSAVES and restored by XRSTORS, in conjunction with the bit field associated with supervisory state component in IA32_XSS. See Chapter 13, “Managing State Using the XSAVE Feature Set,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

36.3.6 Cycle-Accurate Mode

Intel PT can be run in a cycle-accurate mode which enables CYC packets (see Section 36.4.2.14) that provide low-level information in the processor core clock domain. This cycle counter data in CYC packets can be used to compute IPC (Instructions Per Cycle), or to track wall-clock time on a fine-grain level.

To enable cycle-accurate mode packet generation, software should set IA32_RTIT_CTL.CYCEn=1. It is recommended that software also set TSCEn=1 anytime cycle-accurate mode is in use. With this, all CYC-eligible packets will be preceded by a CYC packet, the payload of which indicates the number of core clock cycles since the last CYC packet. In cases where multiple CYC-eligible packets are generated in a single cycle, only a single CYC will be generated before the CYC-eligible packets, otherwise each CYC-eligible packet will be preceded by its own CYC. The CYC-eligible packets are:

- TNT, TIP, TIP.PGE, TIP.PGD, MODE.EXEC, MODE.TSX, PIP, VMCS, OVF, MTC, TSC, PTWRITE, EXSTOP

TSC packets are generated when there is insufficient information to reconstruct wall-clock time, due to tracing being disabled (TriggerEn=0), or power down scenarios like a transition to a deep-sleep MWAIT C-state. In this case, the CYC that is generated along with the TSC will indicate the number of cycles actively tracing (those powered up, with TriggerEn=1) executed between the last CYC packet and the TSC packet. And hence the amount of time spent while tracing is inactive can be inferred from the difference in time between that expected based on the CYC value, and the actual time indicated by the TSC.

Additional CYC packets may be sent stand-alone, so that the processor can ensure that the decoder is aware of the number of cycles that have passed before the internal hardware counter wraps, or is reset due to other micro-architectural condition. There is no guarantee at what intervals these standalone CYC packets will be sent, except that they will be sent before the wrap occurs. An illustration is given below.

Example 36-1. An Illustrative CYC Packet Example

Time (cycles)	Instruction Snapshot	Generated Packets	Comment
x	call %eax	CYC(?), TIP	?Elapsed cycles from the previous CYC unknown
x + 2	call %ebx	CYC(2), TIP	1 byte CYC packet; 2 cycles elapsed from the previous CYC
x + 8	jnz Foo (not taken)	CYC(6)	1 byte CYC packet
x + 9	ret (compressed)		
x + 12	jnz Bar (taken)		
x + 16	ret (uncompressed)	TNT, CYC(8), TIP	1 byte CYC packet
x + 4111		CYC(4095)	2 byte CYC packet
x + 12305		CYC(8194)	3 byte CYC packet
x + 16332	mov cr3, %ebx	CYC(4027), PIP	2 byte CYC packet

36.3.6.1 Cycle Counter

The cycle counter is implemented in hardware (independent of the time stamp counter or performance monitoring counters), and is a simple incrementing counter that does not saturate, but rather wraps. The size of the counter is implementation specific.

The cycle counter is reset to zero any time that TriggerEn is cleared, and when a CYC packet is sent. The cycle counter will continue to count when ContextEn or FilterEn are cleared, and cycle packets will still be generated. It will not count during sleep states that result in Intel PT logic being powered-down, but will count up to the point where clocks are disabled, and resume counting once they are re-enabled.

36.3.6.2 Cycle Packet Semantics

Cycle-accurate mode adheres to the following protocol:

- All packets that precede a CYC packet represent instructions or events that took place before the CYC time.
- All packets that follow a CYC packet represent instructions or events that took place at the same time as, or after, the CYC time.
- The CYC-eligible packet that immediately follows a CYC packet represents an instruction or event that took place at the same time as the CYC time.

These items above give the decoder a means to apply CYC packets to a specific instruction in the assembly stream. Most packets represent a single instruction or event, and hence the CYC packet that precedes each of those packets represents the retirement time of that instruction or event. In the case of TNT packets, up to 6 conditional branches and/or compressed RETs may be contained in the packet. In this case, the preceding CYC packet provides the retirement time of the first branch in the packet. It is possible that multiple branches retired in the same cycle as that first branch in the TNT, but the protocol will not make that obvious. Also note that a MTC packet could be generated in the same cycle as the first JCC in the TNT packet. In this case, the CYC would precede both the MTC and the TNT, and apply to both.

Note that there are times when the cycle counter will stop counting, though cycle-accurate mode is enabled. After any such scenario, a CYC packet followed by TSC packet will be sent. See Section 36.8.3.2 to understand how to interpret the payload values

Multi-packet Instructions or Events

Some operations, such as interrupts or task switches, generate multiple packets. In these cases, multiple CYC packets may be sent for the operation, preceding each CYC-eligible packet in the operation. An example, using a task switch on a software interrupt, is shown below.

Example 36-2. An Example of CYC in the Presence of Multi-Packet Operations

Time (cycles)	Instruction Snapshot	Generated Packets
x	jnz Foo (not taken)	CYC(?)
x + 2	ret (compressed)	
x + 8	jnz Bar (taken)	
x + 9	jmp %eax	TNT, CYC(9), TIP
x + 12	jnz Bar (not taken)	CYC(3)
x + 32	int3 (task gate)	TNT, FUP, CYC(10), PIP, CYC(20), MODE.Exec, TIP

36.3.6.3 Cycle Thresholds

Software can opt to reduce the frequency of cycle packets, a trade-off to save bandwidth and intrusion at the expense of precision. This is done by utilizing a cycle threshold (see Section 36.2.8.2).

IA32_RTIT_CTL.CycThresh indicates to the processor the minimum number of cycles that must pass before the next CYC packet should be sent. If this value is 0, no threshold is used, and CYC packets can be sent every cycle in which a CYC-eligible packet is generated. If this value is greater than 0, the hardware will wait until the associated number of cycles have passed since the last CYC packet before sending another. CPUID provides the threshold options for CycThresh, see Section 36.3.1.

Note that the cycle threshold does not dictate how frequently a CYC packet will be posted, it merely assigns the maximum frequency. If the cycle threshold is 16, a CYC packet can be posted no more frequently than every 16 cycles. However, once that threshold of 16 cycles has passed, it still requires a new CYC-eligible packet to be generated before a CYC will be inserted. Table 36-13 illustrates the threshold behavior.

Table 36-13. An Illustrative CYC Packet Example

Time (cycles)	Instruction Snapshot	Threshold			
		0	16	32	64
x	jmp %eax	CYC, TIP	CYC, TIP	CYC, TIP	CYC, TIP
x + 9	call %ebx	CYC, TIP	TIP	TIP	TIP
x + 15	call %ecx	CYC, TIP	TIP	TIP	TIP
x + 30	jmp %edx	CYC, TIP	CYC, TIP	TIP	TIP
x + 38	mov cr3, %eax	CYC, PIP	PIP	CYC, PIP	PIP
x + 46	jmp [%eax]	CYC, TIP	CYC, TIP	TIP	TIP
x + 64	call %edx	CYC, TIP	CYC, TIP	TIP	CYC, TIP
x + 71	jmp %edx	CYC, TIP	TIP	CYC, TIP	TIP

36.3.7 Decoder Synchronization (PSB+)

The PSB packet (Section 36.4.2.17) serves as a synchronization point for a trace-packet decoder. It is a pattern in the trace log for which the decoder can quickly scan to align packet boundaries. No legal packet combination can result in such a byte sequence. As such, it serves as the starting point for packet decode. To decode a trace log properly, the decoder needs more than simply to be aligned: it needs to know some state and potentially some timing information as well. The decoder should never need to retain any information (e.g., LastIP, call stack, compound packet event) across a PSB; all compound packet events will be completed before a PSB, and any compression state will be reset.

When a PSB packet is generated, it is followed by a PSBEND packet (Section 36.4.2.18). One or more packets may be generated in between those two packets, and these inform the decoder of the current state of the processor. These packets, known collectively as PSB+, should be interpreted as “status only”, since they do not imply any change of state at the time of the PSB, nor are they associated directly with any instruction or event. Thus, the

normal binding and ordering rules that apply to these packets outside of PSB+ can be ignored when these packets are between a PSB and PSBEND. They inform the decoder of the state of the processor at the time of the PSB.

PSB+ can include:

- Timestamp (TSC), if IA32_RTIT_CTL.TSCEn=1.
- Timestamp-MTC Align (TMA), if IA32_RTIT_CTL.TSCEn=1 && IA32_RTIT_CTL.MTCEn=1.
- Paging Information Packet (PIP), if ContextEn=1 and IA32_RTIT_CTL.OS=1. The non-root bit (NR) is set if the logical processor is in VMX non-root operation and the “conceal VMX from PT” VM-execution control is 0.
- VMCS packet, if either the logical is in VMX root operation or the logical processor is in VMX non-root operation and the “conceal VMX from PT” VM-execution control is 0.
- Core Bus Ratio (CBR).
- MODE.TSX, if ContextEn=1 and BranchEn = 1.
- MODE.Exec, if PacketEn=1 or (ContextEn=1 and IA32_RTIT_CTL.EventEn=1).
- Flow Update Packet (FUP), if PacketEn=1.

PSB is generated only when TriggerEn=1; hence PSB+ has the same dependencies. The ordering of packets within PSB+ is not fixed. Timing packets such as CYC and MTC may be generated between PSB and PSBEND, and their meanings are the same as outside PSB+.

A PSB+ can be lost in some scenarios. If IA32_RTIT_STATUS.TriggerEn is cleared just as the PSB threshold is reached, e.g., due to TraceEn being cleared, the PSB+ may not be generated. On processors that support PSB preservation (CPUID.14H.00H:EBX[6] = 1), setting IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1 will ensure that a PSB+ that is pending at the time PT is disabled will be recorded by setting IA32_RTIT_STATUS.PendPSB[6] = 1. A PSB will be inserted, and PendPSB cleared, when PT is later re-enabled while PendPSB = 1.

Note that an overflow can occur during PSB+, and this could cause the PSBEND packet to be lost. For this reason, the OVF packet should also be viewed as terminating PSB+. If IA32_RTIT_STATUS.TriggerEn is cleared just as the PSB threshold is reached, the PSB+ may not be generated. TriggerEn can be cleared by a WRMSR that clears IA32_RTIT_CTL.TraceEn, a VM exit that clears IA32_RTIT_CTL.TraceEn, an #SMI, or any time that either IA32_RTIT_STATUS.Stopped is set (e.g., by a TraceStop or ToPA stop condition) or IA32_RTIT_STATUS.Error is set (e.g., by an Intel PT output error). On processors that support PSB preservation (CPUID.14H.00H:EBX[6] = 1), setting IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1 will ensure that a PSB+ that is pending at the time PT is disabled will be recorded by setting IA32_RTIT_STATUS.PendPSB[6] = 1. A PSB will then be pended when the saved PT context is later restored.

36.3.8 Internal Buffer Overflow

In the rare circumstances when new packets need to be generated but the processor’s dedicated internal buffers are all full, an “internal buffer overflow” occurs. On such an overflow packet generation ceases (as packets would need to enter the processor’s internal buffer) until the overflow resolves. Once resolved, packet generation resumes.

When the buffer overflow is cleared, an OVF packet (Section 36.4.2.16) is generated, and the processor ensures that packets which follow the OVF are not compressed (IP compression or RET compression) against packets that were lost.

If IA32_RTIT_CTL.BranchEn = 1, the OVF packet will be followed by a FUP if the overflow resolves while PacketEn=1. If the overflow resolves while PacketEn = 0 no packet is generated, but a TIP.PGE will naturally be generated later, once PacketEn = 1. The payload of the FUP or TIP.PGE will be the Current IP of the first instruction upon which tracing resumes after the overflow is cleared. If the overflow resolves while PacketEn=1, only timing packets may come between the OVF and the FUP. If the overflow resolves while PacketEn=0, any other packets that are not dependent on PacketEn may come between the OVF and the TIP.PGE.

36.3.8.1 Overflow Impact on Enables

The address comparisons to ADDRn ranges, for IP filtering and TraceStop (Section 36.2.5.3), continue during a buffer overflow, and TriggerEn, ContextEn, and FilterEn may change during a buffer overflow. Like other packets, however, any TIP.PGE or TIP.PGD packets that would have been generated will be lost. Further, IA32_RTIT_STATUS.PacketByteCnt will not increment, since it is only incremented when packets are generated.

If a TraceStop event occurs during the buffer overflow, IA32_RTIT_STATUS.Stopped will still be set, tracing will cease as a result. However, the TraceStop packet, and any TIP.PGD that result from the TraceStop, may be dropped.

36.3.8.2 Overflow Impact on Timing Packets

Any timing packets that are generated during a buffer overflow will be dropped. If only a few MTC packets are dropped, a decoder should be able to detect this by noticing that the time value in the first MTC packet after the buffer overflow incremented by more than one. If the buffer overflow lasted long enough that 256 MTC packets are lost (and thus the MTC packet 'wraps' its 8-bit CTC value), then the decoder may be unable to properly understand the trace. This is not an expected scenario. No CYC packets are generated during overflow, even if the cycle counter wraps.

Note that, if cycle-accurate mode is enabled, the OVF packet will generate a CYC packet. Because the cycle counter counts during overflows, this CYC packet can provide the duration of the overflow. However, there is a risk that the cycle counter wrapped during the overflow, which could render this CYC misleading.

36.3.9 TNT Disable

Software can opt to omit TNT packets from control flow trace (BranchEn=1) by setting IA32_RTIT_CTL.DisTNT[bit 55]. This can dramatically reduce trace size. Results will vary by workload, but trace size reductions of 40-75% are typical, which will have a corresponding reduction in performance overhead and memory bandwidth consumption from Intel PT. However, omitting TNT packets means the decoder is not able to follow the full control flow trace, since conditional branch and compressed RET results won't be known. Thus, TNT Disable should be employed only for usages that do not depend on full control flow trace.

NOTE

To avoid loss of RET results with TNT Disable, software may wish to disable RET compression by setting IA32_RTIT_CTL.DisRETC[bit 11].

36.3.10 Operational Errors

Errors are detected as a result of packet output configuration problems, which can include output alignment issues, ToPA reserved bit violations, or overlapping packet output with restricted memory. See "ToPA Errors" in Section 36.2.7.2 for details on ToPA errors, and Section 36.2.7.4 for details on restricted memory errors. Operational errors are only detected and signaled when TraceEn=1.

When an operational error is detected, tracing is disabled and the error is logged. Specifically, IA32_RTIT_STATUS.Error is set, which will cause IA32_RTIT_STATUS.TriggerEn to be 0. This will disable generation of all packets. Some causes of operational errors may lead to packet bytes being dropped.

It should be noted that the timing of error detection may not be predictable. Errors are signaled when the processor encounters the problematic configuration. This could be as soon as packet generation is enabled but could also be later when the problematic entry or field needs to be used.

Once an error is signaled, software should disable packet generation by clearing TraceEn, diagnose and fix the error condition, and clear IA32_RTIT_STATUS.Error. At this point, packet generation can be re-enabled.

36.4 TRACE PACKETS AND DATA TYPES

This section details the data packets generated by Intel Processor Trace. It is useful for developers writing the interpretation code that will decode the data packets and apply it to the traced source code.

36.4.1 Packet Relationships and Ordering

This section introduces the concept of packet “binding”, which involves determining the IP in a binary disassembly at which the change indicated by a given packet applies. Some packets have the associated IP as the payload (FUP, TIP), while for others the decoder need only search for the next instance of a particular instruction (or instructions) to bind the packet (TNT). However, in many cases, the decoder will need to consider the relationship between packets, and to use this packet context to determine how to bind the packet.

Section 36.4.1.1 below provides detailed descriptions of the packets, including how packets bind to IPs in the disassembly, to other packets, or to nothing at all. Many packets listed are simple to bind, because they are generated in only a few scenarios. Those that require more consideration are typically part of “compound packet events”, such as interrupts, exceptions, and some instructions, where multiple packets are generated by a single operation (instruction or event). These compound packet events frequently begin with a FUP to indicate the source address (if it is not clear from the disassembly), and are concluded by a TIP or TIP.PGD packet that indicates the destination address (if one is provided). In this scenario, the FUP is said to be “coupled” with the TIP packet.

Other packets could be in between the coupled FUP and TIP packet. Timing packets, such as TSC, MTC, CYC, or CBR, could arrive at any time, and hence could intercede in a compound packet event. If an operation changes CR3 or the processor’s mode of execution, a state update packet (i.e., PIP or MODE) is generated. The state changes indicated by these intermediate packets should be applied at the IP of the TIP* packet. A summary of compound packet events is provided in Table 36-14; see Section 36.4.1.1 for more per-packet details and Section 36.7 for more detailed packet generation examples.

Table 36-14. Compound Packet Event Summary

Event Type	Beginning	Middle	End	Comment
Unconditional, uncompressed control-flow transfer	FUP or none	Any combination of PIP, VMCS, MODE.Exec, or none	TIP or TIP.PGD	FUP only for asynchronous events. Order of middle packets may vary. PIP/VMCS/MODE only if the operation modifies the state tracked by these respective packets.
TSX Update	MODE.TSX, and (FUP or none)	None	TIP, TIP.PGD, or none	FUP TIP/TIP.PGD only for TSX abort cases.
Overflow	OVF	PSB, PSBEND, or none	FUP or TIP.PGE	FUP if overflow resolves while ContextEn=1, else TIP.PGE.

36.4.1.1 Packet Blocks

Packet blocks are a means to dump one or more groups of state values. Packet blocks begin with a Block Begin Packet (BBP), which indicates what type of state is held within the block. Following each BBP there may be one or more Block Item Packets (BIPs), which contain the state values. The block is terminated by either a Block End Packet (BEP) or another BBP indicating the start of a new block.

The BIP packet includes an ID value that, when combined with the Type field from the BBP that preceded it, uniquely identifies the state value held in the BIP payload. The size of each BIP packet payload is provided by the Size field in the preceding BBP packet.

Each block type can have up to 32 items defined for it. There is no guarantee, however, that each block of that type will hold all 32 items. For more details on which items to expect, see documentation on the specific block type of interest.

See the BBP packet description (Section 36.4.2.26) for details on packet block generation scenarios.

Packet blocks are entirely generated within an instruction or between instructions, which dictates the types of packets (aside from BIPs) that may be seen within a packet block. Packets that indicate control flow changes, or other indication of instruction completion, cannot be generated within a block. These are listed in the following table. Other packets, including timing packets, may occur between BBP and BEP.

Table 36-15. Packets Forbidden Between BBP and BEP

TNT
TIP, TIP.PGE, TIP.PGD

Table 36-15. Packets Forbidden Between BBP and BEP

MODE.Exec, MODE.TSX
PIP, VMCS
TraceStop
PSB, PSBEND
PTW
MWAIT

It is possible to encounter an internal buffer overflow in the middle of a block. In such a case, it is guaranteed that packet generation will not resume in the middle of a block, and hence the OVF packet terminates the current block. Depending on the duration of the overflow, subsequent blocks may also be lost.

Decoder Implications

When a Block Begin Packet (BBP) is encountered, the decoder will need to decode some packets within the block differently from those outside a block. The Block Item Packet (BIP) header byte has the same encoding as a TNT packet outside of a block, but must be treated as a BIP header (with following payload) within one.

When an OVF packet is encountered, the decoder should treat that as a block ending condition. Packet generation will not resume within a block.

36.4.2 Packet Definitions

The following description of packet definitions are in tabular format. Figure 36-3 explains how to interpret them. Packet bits listed as “RSVD” are not guaranteed to be 0.

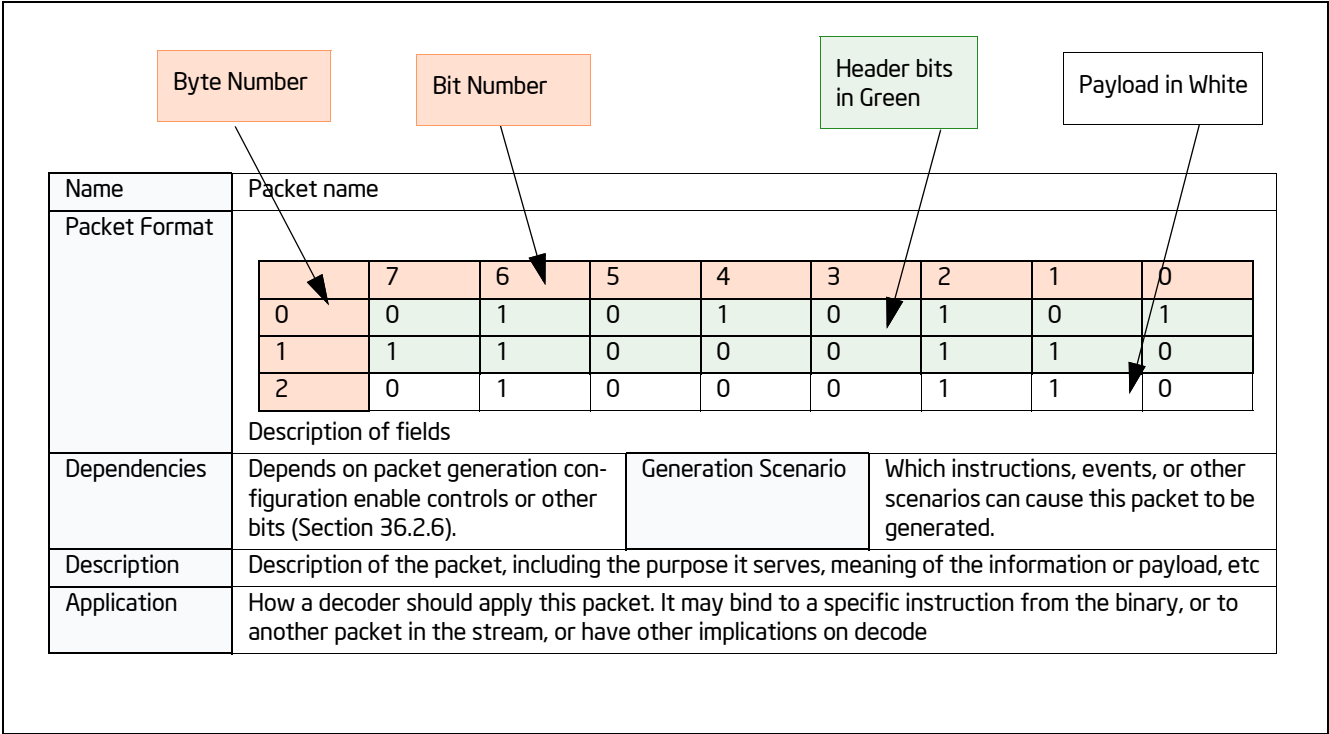


Figure 36-3. Interpreting Tabular Definition of Packet Format

36.4.2.1 Taken/Not-taken (TNT) Packet

Table 36-16. TNT Packet Definition

Name	Taken/Not-taken (TNT) Packet									
Packet Format										
		7	6	5	4	3	2	1	0	
	0	1	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	0	Short TNT
	B1...BN represent the last N conditional branch or compressed RET (Section 36.4.2.2) results, such that B1 is oldest and BN is youngest. The short TNT packet can contain from 1 to 6 TNT bits. The long TNT packet can contain from 1 to 47 TNT bits.									
		7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	1	0	Long TNT
	1	1	0	1	0	0	0	1	1	
	2	B ₄₀	B ₄₁	B ₄₂	B ₄₃	B ₄₄	B ₄₅	B ₄₆	B ₄₇	
	3	B ₃₂	B ₃₃	B ₃₄	B ₃₅	B ₃₆	B ₃₇	B ₃₈	B ₃₉	
	4	B ₂₄	B ₂₅	B ₂₆	B ₂₇	B ₂₈	B ₂₉	B ₃₀	B ₃₁	
	5	B ₁₆	B ₁₇	B ₁₈	B ₁₉	B ₂₀	B ₂₁	B ₂₂	B ₂₃	
	6	B ₈	B ₉	B ₁₀	B ₁₁	B ₁₂	B ₁₃	B ₁₄	B ₁₅	
	7	1	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	B ₇	
	Irrespective of how many TNT bits is in a packet, the last valid TNT bit is followed by a trailing 1, or Stop bit, as shown above. If the TNT packet is not full (fewer than 6 TNT bits for the Short TNT, or fewer than 47 TNT bits for the Long TNT), the Stop bit moves up, and the trailing bits of the packet are filled with 0s. Examples of these “partial TNTs” are shown below. An implementation may choose to use long TNTs, short TNTs, or both.									
		7	6	5	4	3	2	1	0	
	0	0	0	1	B ₁	B ₂	B ₃	B ₄	0	Short TNT
		7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	1	0	Long TNT
	1	1	0	1	0	0	0	1	1	
	2	B ₂₄	B ₂₅	B ₂₆	B ₂₇	B ₂₈	B ₂₉	B ₃₀	B ₃₁	
	3	B ₁₆	B ₁₇	B ₁₈	B ₁₉	B ₂₀	B ₂₁	B ₂₂	B ₂₃	
	4	B ₈	B ₉	B ₁₀	B ₁₁	B ₁₂	B ₁₃	B ₁₄	B ₁₅	
	5	1	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	B ₇	
	6	0	0	0	0	0	0	0	0	
	7	0	0	0	0	0	0	0	0	
Dependencies	PacketEn && ~IA32_R-TIT_CTL.DisTNT			Generation Scenario	On a conditional branch or compressed RET, if it fills the TNT. Also, partial TNTs may be generated at any time, as a result of other packets being generated, or certain micro-architectural conditions occurring, before the TNT is full.					

Table 36-16. TNT Packet Definition (Contd.)

Description	Provides the taken/not-taken results for the last 1..6 (Short TNT) or 1..47 (Long TNT) conditional branches (Jcc, J*CXZ, or LOOP) or compressed RETs (Section 36.4.2.2). The TNT payload bits should be interpreted as follows: 1 indicates a taken conditional branch, or a compressed RET 0 indicates a not-taken conditional branch TNT payload bits are stored internal to the processor in a TNT buffer, until either the buffer is filled or another packet is to be generated. In either case a TNT packet holding the buffered bits will be emitted, and the TNT buffer will be marked as empty.
Application	Each valid payload bit (that is, bits between the header bits and the trailing Stop bit) applies to an upcoming conditional branch or RET instruction. Once a decoder consumes a TNT packet with N valid payload bits, these bits should be applied to (and hence provide the destination for) the next N conditional branches or RETs.

36.4.2.2 Target IP (TIP) Packet

Table 36-17. TIP Packet Definition

Name	Target IP (TIP) Packet								
Packet Format		7	6	5	4	3	2	1	0
	0	IPBytes			0	1	1	0	1
	1	TargetIP[7:0]							
	2	TargetIP[15:8]							
	3	TargetIP[23:16]							
	4	TargetIP[31:24]							
	5	TargetIP[39:32]							
	6	TargetIP[47:40]							
	7	TargetIP[55:48]							
	8	TargetIP[63:56]							
Dependencies	PacketEn		Generation Scenario		Indirect branch (including un-compressed RET), far branch, interrupt, exception, INIT, SIPI, VM exit, VM entry, TSX abort, EENTER, EEXIT, ERESUME, AEX ¹ .				
Description	Provides the target for some control flow transfers								
Application	Anytime a TIP is encountered, it indicates that control was transferred to the IP provided in the payload.								
	The source of this control flow change, and hence the IP or instruction to which it binds, depends on the packets that precede the TIP. If a TIP is encountered and all preceding packets have already been bound, then the TIP will apply to the upcoming indirect branch, far branch, or VMRESUME. However, if there was a preceding FUP that remains unbound, it will bind to the TIP. Here, the TIP provides the target of an asynchronous event or TSX abort that occurred at the IP given in the FUP payload. Note that there may be other packets, in addition to the FUP, which will bind to the TIP packet. See the packet application descriptions for other packets for details.								

NOTES:

1. EENTER, EEXIT, ERESUME, AEX would be possible only for a debug enclave.

IP Compression

The IP payload in a TIP, FUP, TIP.PGE, or TIP.PGD packet can vary in size, based on the mode of execution, and the use of IP compression. IP compression is an optional compression technique the processor may choose to employ to reduce bandwidth. With IP compression, the IP to be represented in the payload is compared with the last IP sent out, via any of FUP, TIP, TIP.PGE, or TIP.PGD. If that previous IP had the same upper (most significant) address bytes, those matching bytes may be suppressed in the current packet. The processor maintains an internal state of the “Last IP” that was encoded in trace packets, thus the decoder will need to keep track of the “Last IP” state in

software, to match fidelity with packets generated by hardware. “Last IP” is initialized to zero, hence if the first IP in the trace may be compressed if the upper bytes are zeroes.

The “IPBytes” field of the IP packets (FUP, TIP, TIP.PGE, TIP.PGD) serves to indicate how many bytes of payload are provided, and how the decoder should fill in any suppressed bytes. The algorithm for reconstructing the IP for a TIP/FUP packet is shown in the table below.

Table 36-18. FUP/TIP IP Reconstruction

IPBytes	Uncompressed IP Value							
	63:56	55:48	47:40	39:32	31:24	23:16	15:8	7:0
000b	None, IP is out of context							
001b	Last IP[63:16]						IP Payload[15:0]	
010b	Last IP[63:32]				IP Payload[31:0]			
011b	IP Payload[47] extended		IP Payload[47:0]					
100b	Last IP [63:48]		IP Payload[47:0]					
101b	Reserved							
110b	IP Payload[63:0]							
111b	Reserved							

The processor-internal Last IP state is guaranteed to be reset to zero when a PSB is sent out. This means that the IP that follows the PSB with either be un-compressed (011b or 110b, see Table 36-18), or compressed against zero.

At times, “IPbytes” will have a value of 0. As shown above, this does not mean that the IP payload matches the full address of the last IP, but rather that the IP for this packet was suppressed. This is used for cases where the IP that applies to the packet is out of context. An example is the TIP.PGD sent on a SYSCALL, when tracing only USR code. In that case, no TargetIP will be included in the packet, since that would expose an instruction point at CPL = 0. When the IP payload is suppressed in this manner, Last IP is not cleared, and instead refers to the last IP packet with a non-zero IPBytes field.

On processors that support a maximum linear address size of 32 bits, IP payloads may never exceed 32 bits (IPBytes <= 010b).

Indirect Transfer Compression for Returns (RET)

In addition to IP compression, TIP packets for near return (RET) instructions can also be compressed. If the RET target matches the next IP of the corresponding CALL, then the TIP packet is unneeded, since the decoder can deduce the target IP by maintaining a CALL/RET stack of its own.

When a RET is compressed, a Taken indication is added to the TNT buffer. Because the RET generates no TIP packet, it also does not update the internal Last IP value, and thus the decoder should treat it the same way. If the RET is not compressed, it will generate a TIP packet (just like when RET compression is disabled, via IA32_R-TIT_CTL.DisRETC).

A CALL/RET stack can be maintained by the decoder by doing the following:

1. Allocate space to store 64 RET targets.
2. For near CALLs, push the Next IP onto the stack. Once the stack is full, new CALLs will force the oldest entry off the end of the stack, such that only the youngest 64 entries are stored. Note that this excludes zero-length CALLs, which are direct near CALLs with displacement zero (to the next IP). These CALLs typically don't have matching RETs.
3. For near RETs, pop the top (youngest) entry off the stack. This will be the expected target of the RET.

In cases where a RET is compressed, the RET target is guaranteed to match the expected target from 3) above. If the target is not compressed, a TIP packet will be generated with the RET target, which may differ from the expected target in some cases.

The hardware ensures that packets read by the decoder will always have seen the CALL that corresponds to any compressed RET. The processor will never compress a RET across a PSB, a buffer overflow, or scenario where PacketEn=0. This means that a RET whose corresponding CALL executed while PacketEn=0, or before the last PSB, etc., will not be compressed.

If the CALL/RET stack is manipulated or corrupted by software, and thereby causes a RET to transfer control to a target that is inconsistent with the CALL/RET stack, then the RET will not be compressed, and will produce a TIP packet. This can happen, for example, if software executes a PUSH instruction to push a target onto the stack, and a later RET uses this target.

For processors that employ deferred TIPs (Section 36.4.2.3), an uncompressed RET will not be deferred, and hence will force out any accumulated TNTs or TIPs. This serves to avoid ambiguity, and make clear to the decoder whether the near RET was compressed, and hence a bit in the in-progress TNT should be consumed, or uncompressed, in which case there will be no in-progress TNT and thus a TIP should be consumed.

Note that in the unlikely case that a RET executes in a different execution mode than the associated CALL, the decoder will need to model the same behavior with its CALL stack. For instance, if a CALL executes in 64-bit mode, a 64-bit IP value will be pushed onto the software stack. If the corresponding RET executes in 32-bit mode, then only the lower 32 target bits will be popped off of the stack, which may mean that the RET does not go to the CALL's Next IP. This is architecturally correct behavior, and this RET could be compressed, thus the decoder should match this behavior.

36.4.2.3 Deferred TIPs

The processor may opt to defer sending out the TNT when TIPs are generated. Thus, rather than sending a partial TNT followed by a TIP, both packets will be deferred while the TNT accumulates more Jcc/RET results. Any number of TIP packets may be accumulated this way, such that only once the TNT is filled, or once another packet (e.g., FUP) is generated, the TNT will be sent, followed by all the deferred TIP packets, and finally terminated by the other packet(s) that forced out the TNT and TIP packets. Generation of many other packets (see list below) will force out the TNT and any accumulated TIP packets. This is an optional optimization in hardware to reduce the bandwidth consumption, and hence the performance impact, incurred by tracing.

Table 36-19. TNT Examples with Deferred TIPs

Code Flow	Packets, Non-Deferred TIPS	Packets, Deferred TIPS
0x1000 cmp %rcx, 0 0x1004 jnz Foo // not-taken 0x1008 jmp %rdx	TNT(0b0), TIP(0x1308)	
0x1308 cmp %rcx, 1 0x130c jnz Bar // not-taken 0x1310 cmp %rcx, 2 0x1314 jnz Baz // taken 0x1500 cmp %eax, 7 0x1504 jg Exit // not-taken 0x1508 jmp %r15	TNT(0b010), TIP(0x1100)	
0x1100 cmp %rbx, 1 0x1104 jg Start // not-taken 0x1108 add %rcx, %eax 0x110c ... // an asynchronous Interrupt arrives INThandler: 0xcc00 pop %rdx	TNT(0b0), FUP(0x110c), TIP(0xcc00)	TNT(0b00100), TIP(0x1308), TIP(0x1100), FUP(0x110c), TIP(0xcc00)

36.4.2.4 Packet Generation Enable (TIP.PGE) Packet

Table 36-20. TIP.PGE Packet Definition

Name	Target IP - Packet Generation Enable (TIP.PGE) Packet																																																																																																	
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td colspan="3">IPBytes</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td colspan="8">TargetIP[7:0]</td></tr><tr><td>2</td><td colspan="8">TargetIP[15:8]</td></tr><tr><td>3</td><td colspan="8">TargetIP[23:16]</td></tr><tr><td>4</td><td colspan="8">TargetIP[31:24]</td></tr><tr><td>5</td><td colspan="8">TargetIP[39:32]</td></tr><tr><td>6</td><td colspan="8">TargetIP[47:40]</td></tr><tr><td>7</td><td colspan="8">TargetIP[55:48]</td></tr><tr><td>8</td><td colspan="8">TargetIP[63:56]</td></tr></table>									7	6	5	4	3	2	1	0	0	IPBytes			1	0	0	0	1	1	TargetIP[7:0]								2	TargetIP[15:8]								3	TargetIP[23:16]								4	TargetIP[31:24]								5	TargetIP[39:32]								6	TargetIP[47:40]								7	TargetIP[55:48]								8	TargetIP[63:56]							
	7	6	5	4	3	2	1	0																																																																																										
0	IPBytes			1	0	0	0	1																																																																																										
1	TargetIP[7:0]																																																																																																	
2	TargetIP[15:8]																																																																																																	
3	TargetIP[23:16]																																																																																																	
4	TargetIP[31:24]																																																																																																	
5	TargetIP[39:32]																																																																																																	
6	TargetIP[47:40]																																																																																																	
7	TargetIP[55:48]																																																																																																	
8	TargetIP[63:56]																																																																																																	
Dependencies	PacketEn transitions to 1			Generation Scenario	Any branch instruction, control flow transfer, or MOV CR3 that sets PacketEn, a WRMSR that enables packet generation and sets PacketEn																																																																																													
Description	Indicates that PacketEn has transitioned to 1. It provides the IP at which the tracing begins. This can occur due to any of the enables that comprise PacketEn transitioning from 0 to 1, as long as all the others are asserted. Examples: ▪ TriggerEn: This is set on software write to set IA32_RTIT_CTL.TraceEn as long as the Stopped and Error bits in IA32_RTIT_STATUS are clear. The IP payload will be the Next IP of the WRMSR. ▪ FilterEn: This is set when software jumps into the tracing region. This region is defined by enabling IP filtering in IA32_RTIT_CTL.ADDRn_CFG, and defining the range in IA32_RTIT_ADDRn_[AB], see. Section 36.2.5.3. The IP payload will be the target of the branch. ▪ ContextEn: This is set on a CPL change, a CR3 write or any other means of changing ContextEn. The IP payload will be the Next IP of the instruction that changes context if it is not a branch, otherwise it will be the target of the branch.																																																																																																	
Application	TIP.PGE packets bind to the instruction at the IP given in the payload.																																																																																																	

36.4.2.5 Packet Generation Disable (TIP.PGD) Packet

Table 36-21. TIP.PGD Packet Definition

Name	Target IP - Packet Generation Disable (TIP.PGD) Packet																																																																																																	
Packet Format	<table border="1"> <tr> <th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr> <tr> <td>0</td><td colspan="3">IPBytes</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td colspan="8">TargetIP[7:0]</td></tr> <tr> <td>2</td><td colspan="8">TargetIP[15:8]</td></tr> <tr> <td>3</td><td colspan="8">TargetIP[23:16]</td></tr> <tr> <td>4</td><td colspan="8">TargetIP[31:24]</td></tr> <tr> <td>5</td><td colspan="8">TargetIP[39:32]</td></tr> <tr> <td>6</td><td colspan="8">TargetIP[47:40]</td></tr> <tr> <td>7</td><td colspan="8">TargetIP[55:48]</td></tr> <tr> <td>8</td><td colspan="8">TargetIP[63:56]</td></tr> </table>									7	6	5	4	3	2	1	0	0	IPBytes			0	0	0	0	1	1	TargetIP[7:0]								2	TargetIP[15:8]								3	TargetIP[23:16]								4	TargetIP[31:24]								5	TargetIP[39:32]								6	TargetIP[47:40]								7	TargetIP[55:48]								8	TargetIP[63:56]							
	7	6	5	4	3	2	1	0																																																																																										
0	IPBytes			0	0	0	0	1																																																																																										
1	TargetIP[7:0]																																																																																																	
2	TargetIP[15:8]																																																																																																	
3	TargetIP[23:16]																																																																																																	
4	TargetIP[31:24]																																																																																																	
5	TargetIP[39:32]																																																																																																	
6	TargetIP[47:40]																																																																																																	
7	TargetIP[55:48]																																																																																																	
8	TargetIP[63:56]																																																																																																	

Table 36-21. TIP.PGD Packet Definition

Dependencies	PacketEn transitions to 0	Generation Scenario	Any branch instruction, control flow transfer, or MOV CR3 that clears PacketEn, a WRMSR that disables packet generation and clears PacketEn
Description	Indicates that PacketEn has transitioned to 0. It will include the IP at which the tracing ends, unless ContextEn= 0 or TraceEn=0 at the conclusion of the instruction or event that cleared PacketEn. PacketEn can be cleared due to any of the enables that comprise PacketEn transitioning from 1 to 0. Examples: ▪ TriggerEn: This is cleared on software write to clear IA32_RTIT_CTL.TraceEn, or when IA32_RTIT_STATUS.Stopped is set, or on operational error. The IP payload will be suppressed in this case, and the “IPBytes” field will have the value 0. ▪ FilterEn: This is cleared when software jumps out of the tracing region. This region is defined by enabling IP filtering in IA32_RTIT_CTL.ADDRn_CFG, and defining the range in IA32_RTIT_ADDRn_[AB], see. Section 36.2.5.3. The IP payload will depend on the type of the branch. For conditional branches, the payload is suppressed (IPBytes = 0), and in this case the destination can be inferred from the disassembly. For any other type of branch, the IP payload will be the target of the branch. ▪ ContextEn: This can happen on a CPL change, a CR3 write or any other means of changing ContextEn. See Section 36.2.5.3 for details. In this case, when ContextEn is cleared, there will be no IP payload. The “IPBytes” field will have value 0. Note that, in cases where a branch that would normally produce a TIP packet (i.e., far transfer, indirect branch, interrupt, etc) or TNT update (conditional branch or compressed RT) causes PacketEn to transition from 1 to 0, the TIP or TNT bit will be replaced with TIP.PGD. The payload of the TIP.PGD will be the target of the branch, unless the result of the instruction causes TraceEn or ContextEn to be cleared (ie, SYSCALL when IA32_RTIT_CTL.OS=0, In the case where a conditional branch clears FilterEn and hence PacketEn, there will be no TNT bit for this branch, replaced instead by the TIP.PGD.		
Application	TIP.PGD can be produced by any branch instructions, as well as some non-branch instructions, that clear PacketEn. When produced by a branch, it replaces any TIP or TNT update that the branch would normally produce. In cases where there is an unbound FUP preceding the TIP.PGD, then the TIP.PGD is part of compound operation (i.e., asynchronous event or TSX abort) which cleared PacketEn. For most such cases, the TIP.PGD is simply replacing a TIP, and should be treated the same way. The TIP.PGD may or may not have an IP payload, depending on whether the operation cleared ContextEn. If there is not an associated FUP, the binding will depend on whether there is an IP payload. If there is an IP payload, then the TIP.PGD should be applied to either the next direct branch whose target matches the TIP.PGD payload, or the next branch that would normally generate a TIP or TNT packet. If there is no IP payload, then the TIP.PGD should apply to the next branch or MOV CR3 instruction.		

36.4.2.6 Flow Update (FUP) Packet

Table 36-22. FUP Packet Definition

Name	Flow Update (FUP) Packet																																																																																																	
Packet Format	<table border="1"> <thead> <tr> <th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr> </thead> <tbody> <tr> <td>0</td><td colspan="3">IPBytes</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td colspan="8">IP[7:0]</td></tr> <tr> <td>2</td><td colspan="8">IP[15:8]</td></tr> <tr> <td>3</td><td colspan="8">IP[23:16]</td></tr> <tr> <td>4</td><td colspan="8">IP[31:24]</td></tr> <tr> <td>5</td><td colspan="8">IP[39:32]</td></tr> <tr> <td>6</td><td colspan="8">IP[47:40]</td></tr> <tr> <td>7</td><td colspan="8">IP[55:48]</td></tr> <tr> <td>8</td><td colspan="8">IP[63:56]</td></tr> </tbody> </table>									7	6	5	4	3	2	1	0	0	IPBytes			1	1	1	0	1	1	IP[7:0]								2	IP[15:8]								3	IP[23:16]								4	IP[31:24]								5	IP[39:32]								6	IP[47:40]								7	IP[55:48]								8	IP[63:56]							
	7	6	5	4	3	2	1	0																																																																																										
0	IPBytes			1	1	1	0	1																																																																																										
1	IP[7:0]																																																																																																	
2	IP[15:8]																																																																																																	
3	IP[23:16]																																																																																																	
4	IP[31:24]																																																																																																	
5	IP[39:32]																																																																																																	
6	IP[47:40]																																																																																																	
7	IP[55:48]																																																																																																	
8	IP[63:56]																																																																																																	

Table 36-22. FUP Packet Definition

Dependencies	TriggerEn && ContextEn. (Typically depends on BranchEn and FilterEn as well, see Section 36.2.5, Section 36.4.2.21, and Section 36.4.2.22 for details.)	Generation Scenario	Asynchronous Events (interrupts, exceptions, INIT, SIPI, SMI, VM exit, #MC), PSB+, XBEGIN, XEND, XABORT, XACQUIRE, XRELEASE, EENTER, EEXIT, ERESUME, EEE, AEX, ¹ INTO, INT1, INT3, INT <i>n</i> , a WRMSR that disables packet generation. In addition, when FRED transitions are enabled, SYSCALL and SYSENTER.
Description	Provides the source address for asynchronous events, and some other instructions. Is never sent alone, always sent with an associated TIP or MODE packet, and potentially others.		
Application	FUP packets provide the IP to which they bind. However, they are never standalone, but are coupled with other packets. In TSX cases, the FUP is immediately preceded by a MODE.TSX, which binds to the same IP. A TIP will follow only in the case of TSX aborts, see Section 36.4.2.8 for details. Otherwise, FUPs are part of compound packet events (see Section 36.4.1). In these compound cases, the FUP provides the source IP for an instruction or event, while a following TIP (or TIP.PGD) packet will provide the destination IP. Other packets may be included in the compound event between the FUP and TIP.		

NOTES:

1. EENTER, EEXIT, ERESUME, EEE, AEX apply only if Intel Software Guard Extensions is supported.

FUP IP Payload

Flow Update Packet gives the source address of an instruction when it is needed. In general, branch instructions do not need a FUP, because the source address is clear from the disassembly. For asynchronous events, however, the source address cannot be inferred from the source, and hence a FUP will be sent. Table 36-23 illustrates cases where FUPs are sent, and which IP can be expected in those cases.

Table 36-23. FUP Cases and IP Payload

Event	Flow Update IP	Comment
External Interrupt, NMI/SMI, Traps, Machine Check (trap-like), INIT/SIPI	Address of next instruction (Next IP) that would have been executed	Functionally, this matches the LBR FROM field value and also the EIP value which is saved onto the stack.
Exceptions/Faults, Machine check (fault-like)	Address of the instruction which took the exception/fault (Current IP)	This matches the similar functionality of LBR FROM field value and also the EIP value which is saved onto the stack.
Software Interrupt (and also SYSCALL and SYSENTER when FRED transitions are enabled)	Address of the software interrupt instruction (Current IP)	This matches the similar functionality of LBR FROM field value, but does not match the EIP value which is saved onto the stack (Next Linear Instruction Pointer - NLIP).
EENTER, EEXIT, ERESUME, Enclave Exiting Event (EEE), AEX ¹	Current IP of the instruction	This matches the LBR FROM field value and also the EIP value which is saved onto the stack.
XACQUIRE	Address of the X* instruction	
XRELEASE, XBEGIN, XEND, XABORT, other transactional abort	Current IP	
#SMI	IP that is saved into SMRAM	
WRMSR that clears TraceEn, PSB+	Current IP	

NOTES:

1. Information on EENTER, EEXIT, ERESUME, EEE, Asynchronous Enclave eXit (AEX) can be found in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D.

On a canonical fault due to sequentially fetching an instruction in non-canonical space (as opposed to jumping to non-canonical space), the IP of the fault (and thus the payload of the FUP) will be a non-canonical address. This is consistent with what is pushed on the stack for such faulting cases.

If there are post-commit task switch faults, the IP value of the FUP will be the original IP when the task switch started. This is the same value as would be seen in the LBR_FROM field. But it is a different value as is saved on the stack or VMCS.

36.4.2.7 Paging Information (PIP) Packet

Table 36-24. PIP Packet Definition

Name	Paging Information (PIP) Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	0	1	0	0	0	0	1	1
	2	CR3[11:5] or 0							RSVD/NR
	3	CR3[19:12]							
	4	CR3[27:20]							
	5	CR3[35:28]							
	6	CR3[43:36]							
	7	CR3[51:44]							
Dependencies	TriggerEn && ContextEn && IA32_R-TIT_CTL.OS			Generation Scenario		MOV CR3, Task switch, INIT, SIPI, PSB+, VM exit, VM entry			
Description	The CR3 payload shown includes only the address portion of the CR3 value. For PAE paging, CR3[11:5] are thus included. For other paging modes (32-bit and 4-level paging¹), these bits are 0. This packet holds the CR3 address value. It will be generated on operations that modify CR3: ▪ MOV CR3 operation ▪ Task Switch ▪ INIT and SIPI ▪ VM exit, if “conceal VMX from PT” VM-exit control is 0 (see Section 36.5.1) ▪ VM entry, if “conceal VMX from PT” VM-entry control is 0 PIPs are not generated, despite changes to CR3, on SMI and RSM. This is due to the special behavior on these operations, see Section 36.2.9.3 for details. Note that, for some cases of task switch where CR3 is not modified, no PIP will be produced. The purpose of the PIP is to indicate to the decoder which application is running, so that it can apply the proper binaries to the linear addresses that are being traced. The PIP packet contains the new CR3 value when CR3 is written. PIPs generated by VM entries set the NR bit. PIPs generated in VMX non-root operation set the NR bit if the “conceal VMX from PT” VM-execution control is 0 (see Section 36.5.1). All other PIPs clear the NR bit.								
Application	The purpose of the PIP packet is to help the decoder uniquely identify what software is running at any given time. When a PIP is encountered, a decoder should do the following: 1) If there was a prior unbound FUP (that is, a FUP not preceded by a packet such as MODE.TSX that consumes it, and it hence pairs with a TIP that has not yet been seen), then this PIP is part of a compound packet event (Section 36.4.1). Find the ending TIP and apply the new CR3/NR values to the TIP payload IP. 2) Otherwise, look for the next MOV CR3, far branch, or VMRESUME/VMLAUNCH in the disassembly, and apply the new CR3 to the next (or target) IP. For examples of the packets generated by these flows, see Section 36.7.								

NOTES:

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

36.4.2.8 MODE Packets

MODE packets keep the decoder informed of various processor modes about which it needs to know in order to properly manage the packet output, or to properly disassemble the associated binaries. MODE packets include a header and a mode byte, as shown below.

Table 36-25. General Form of MODE Packets

	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1
1	Leaf ID			Mode				

The MODE Leaf ID indicates which set of mode bits are held in the lower bits.

MODE.Exec Packet

Table 36-26. MODE.Exec Packet Definition

Name	MODE.Exec Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	1	0	0	1	1	0	0	1
	1	0	0	0	Reserved		IF	CS.D	(CS.L & LMA)
	MODE Leaf ID is '000.								
Dependencies	TriggerEn && ContextEn && FilterEn	Generation Scenario	Any operation that changes the CS.L, CS.D, or EFER.LMA, if IA32_R- TIT_CTL.BranchEn=1. Any operation that changes RFLAGS.IF, if IA32_RTIT_CTL.EventEn=1. Any TIP.PGE scenario, such that any of the mode bits tracked may have changed since the last MODE.Exec.						

Table 36-26. MODE.Exec Packet Definition

Description	Indicates whether software is in 16, 32, or 64-bit mode, by providing the CS.D and (CS.L & IA32_EFER.LMA) values. Essential for the decoder to properly disassemble the associated binary. Further, if CPUID.14H.00H:EBX[7]=1 ("Event Trace Support"), it indicates when interrupts are masked by providing the RFLAGS.IF value.		
	MODE.Exec is sent at the time of a mode change, if dependencies are met at the time, otherwise it is sent when tracing resumes. In the former case, the MODE.Exec packet is generated along with other packets that result from the operation that changes the mode, and is guaranteed to be followed by a TIP or TIP.PGE for branch operations, or a FUP for non-branch operations (CLI, STI, or POPF if EventEn=1). In cases where the mode changes while filtering dependencies are not met, the processor ensures that the decoder doesn't lose track of the mode by sending any needed MODE.Exec once tracing resumes (preceding the TIP.PGE, if BranchEn=1). The processor may opt to suppress the MODE.Exec when tracing resumes if the mode matches that of the last MODE.Exec packet. MODE.Exec packets are generated on CS.L, CS.D, or EFER.LMA changes only if control flow tracing is enabled (BranchEn=1). This is essential for the decoder to properly disassemble the associated binary.		
	CS.D	(CS.L & IA32_EFER.LMA)	Addressing Mode
	1	1	N/A
	0	1	64-bit mode
	1	0	32-bit mode
	0	0	16-bit mode
	MODE.Exec packets are generated on interrupt flag (RFLAGS.IF) changes only if event tracing is enabled (EventEn=1). The IF field in MODE.Exec packets is populated only if EventEn=1 (IF = EFLAGS.IF & EventEn).		
Application	MODE.Exec always precedes an IP packet (TIP, TIP.PGE, or FUP). The mode change applies to the IP address in the payload of the IP packet. When MODE.Exec is followed by a FUP, it is a stand-alone FUP and should be consumed by the MODE.Exec.		

MODE.TSX Packet**Table 36-27. MODE.TSX Packet Definition**

Name	MODE.TSX Packet																																		
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>TXAbort</td><td>InTX</td></tr></table>									7	6	5	4	3	2	1	0	0	1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	TXAbort	InTX
	7	6	5	4	3	2	1	0																											
0	1	0	0	1	1	0	0	1																											
1	0	0	1	0	0	0	TXAbort	InTX																											
Dependencies	TriggerEn && ContextEn			Generation Scenario	XBEGIN, XEND, XABORT, XACQUIRE, XRELEASE, if InTX changes, Asynchronous TSX Abort, PSB+																														
Description	Indicates when a TSX transaction (either HLE or RTM) begins, commits, or aborts. Instructions executed transactionally will be “rolled back” if the transaction is aborted.																																		
	TXAbort	InTX	Implication																																
	1	1	N/A																																
	0	1	Transaction begins, or executing transactionally																																
	1	0	Transaction aborted																																
	0	0	Transaction committed, or not executing transactionally																																

Table 36-27. MODE.TSX Packet Definition

Application	If PacketEn=1, MODE.TSX always immediately precedes a FUP. If the TXAbort bit is zero, then the mode change applies to the IP address in the payload of the FUP. If TXAbort=1, then the FUP will be followed by a TIP, and the mode change will apply to the IP address in the payload of the TIP. MODE.TSX packets may be generated when PacketEn=0, due to FilterEn=0. In this case, only the last MODE.TSX generated before TIP.PGE need be applied.

36.4.2.9 TraceStop Packet

Table 36-28. TraceStop Packet Definition

Name	TraceStop Packet																																		
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	1
	7	6	5	4	3	2	1	0																											
0	0	0	0	0	0	0	1	0																											
1	1	0	0	0	0	0	1	1																											
Dependencies	TriggerEn && ContextEn		Generation Scenario	Taken branch with target in TraceStop IP region, MOV CR3 in TraceStop IP region, or WRMSR that sets TraceEn in TraceStop IP region.																															
Description	Indicates when software has entered a user-configured TraceStop region. When the IP matches a TraceStop range while ContextEn and TriggerEn are set, a TraceStop action occurs. This disables tracing by setting IA32_RTIT_STATUS.Stopped, thereby clearing TriggerEn, and causes a TraceStop packet to be generated. The TraceStop action also forces FilterEn to 0. Note that TraceStop may not force a flush of internally buffered packets, and thus trace packet generation should still be manually disabled by clearing IA32_RTIT_CTL.TraceEn before examining output. See Section 36.2.5.3 for more details.																																		
Application	If TraceStop follows a TIP.PGD (before the next TIP.PGE), then it was triggered either by the instruction that cleared PacketEn, or it was triggered by some later instruction that executed while FilterEn=0. In either case, the TraceStop can be applied at the IP of the TIP.PGD (if any). If TraceStop follows a TIP.PGE (before the next TIP.PGD), it should be applied at the last known IP.																																		

36.4.2.10 Core:Bus Ratio (CBR) Packet

Table 36-29. CBR Packet Definition

Name	Core:Bus Ratio (CBR) Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	0	0	0	0	0	0	1	1
	2	Core:Bus Ratio							
	3	Reserved							
Dependencies	TriggerEn			Generation Scenario	After any frequency change, on C-state wake up, PSB+, and after enabling trace packet generation.				
Description	Indicates the core:bus ratio of the processor core. Useful for correlating wall-clock time and cycle time.								
Application	The CBR packet indicates the point in the trace when a frequency transition has occurred. On some implementations, software execution will continue during transitions to a new frequency, while on others software execution ceases during frequency transitions. There is not a precise IP provided, to which to bind the CBR packet.								

36.4.2.11 Timestamp Counter (TSC) Packet

Table 36-30. TSC Packet Definition

Name	Timestamp Counter (TSC) Packet																																																																																								
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td colspan="8">SW TSC[7:0]</td></tr><tr><td>2</td><td colspan="8">SW TSC[15:8]</td></tr><tr><td>3</td><td colspan="8">SW TSC[23:16]</td></tr><tr><td>4</td><td colspan="8">SW TSC[31:24]</td></tr><tr><td>5</td><td colspan="8">SW TSC[39:32]</td></tr><tr><td>6</td><td colspan="8">SW TSC[47:40]</td></tr><tr><td>7</td><td colspan="8">SW TSC[55:48]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	1	1	0	0	1	1	SW TSC[7:0]								2	SW TSC[15:8]								3	SW TSC[23:16]								4	SW TSC[31:24]								5	SW TSC[39:32]								6	SW TSC[47:40]								7	SW TSC[55:48]							
	7	6	5	4	3	2	1	0																																																																																	
0	0	0	0	1	1	0	0	1																																																																																	
1	SW TSC[7:0]																																																																																								
2	SW TSC[15:8]																																																																																								
3	SW TSC[23:16]																																																																																								
4	SW TSC[31:24]																																																																																								
5	SW TSC[39:32]																																																																																								
6	SW TSC[47:40]																																																																																								
7	SW TSC[55:48]																																																																																								
Dependencies	IA32_RTIT_CTL.TSCEn && TriggerEn		Generation Scenario	Sent after any event that causes the processor clocks or Intel PT timing packets (such as MTC or CYC) to stop. This may include P-state changes, wake from C-state, or clock modulation. Also on transition of TraceEn from 0 to 1.																																																																																					
Description	When enabled by software, a TSC packet provides the lower 7 bytes of the current TSC value, as returned by the RDTSC instruction. This may be useful for tracking wall-clock time, and synchronizing the packets in the log with other timestamped logs.																																																																																								
Application	TSC packet provides a wall-clock proxy of the event which generated it (packet generation enable, sleep state wake, etc). In all cases, TSC does not precisely indicate the time of any control flow packets; however, all preceding packets represent instructions that executed before the indicated TSC time, and all subsequent packets represent instructions that executed after it. There is not a precise IP to which to bind the TSC packet.																																																																																								

36.4.2.12 Mini Time Counter (MTC) Packet

Table 36-31. MTC Packet Definition

Name	Mini time Counter (MTC) Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	0	1	0	1	1	0	0	1
	1	CTC[N+7:N]							
Dependencies	IA32_RTIT_CTL.MTCEn && TriggerEn		Generation Scenario		Periodic, based on the core crystal clock, or Always Running Timer (ART).				

Table 36-31. MTC Packet Definition

Description	When enabled by software, an MTC packet provides a periodic indication of wall-clock time. The 8-bit CTC (Common Timestamp Copy) payload value is set to (ART >> N) & FFH. The frequency of the ART is related to the Maximum Non-Turbo frequency, and the ratio can be determined from CPUID.15H, as described in Section 36.8.3. Software can select the threshold N, which determines the MTC frequency by setting the IA32_RTIT_CTL.MTCFreq field (see Section 36.2.8.2) to a supported value using the lookup enumerated by CPUID (see Section 36.3.1). See Section 36.8.3 for details on how to use the MTC payload to track TSC time. MTC provides 8 bits from the ART, starting with the bit selected by MTCFreq to dictate the frequency of the packet. Whenever that 8-bit range being watched changes, an MTC packet will be sent out with the new value of that 8-bit range. This allows the decoder to keep track of how much wall-clock time has elapsed since the last TSC packet was sent, by keeping track of how many MTC packets were sent and what their value was. The decoder can infer the truncated bits, CTC[N-1:0], are 0 at the time of the MTC packet. There are cases in which MTC packet can be dropped, due to overflow or other micro-architectural conditions. The decoder should be able to recover from such cases by checking the 8-bit payload of the next MTC packet, to determine how many MTC packets were dropped. It is not expected that >256 consecutive MTC packets should ever be dropped.
Application	MTC does not precisely indicate the time of any other packet, nor does it bind to any IP. However, all preceding packets represent instructions or events that executed before the indicated ART time, and all subsequent packets represent instructions that executed after, or at the same time as, the ART time.

36.4.2.13 TSC/MTC Alignment (TMA) Packet

Table 36-32. TMA Packet Definition

Name	TSC/MTC Alignment (TMA) Packet																																																																															
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>2</td><td colspan="8">CTC[7:0]</td></tr><tr><td>3</td><td colspan="8">CTC[15:8]</td></tr><tr><td>4</td><td colspan="7">Reserved</td><td>0</td></tr><tr><td>5</td><td colspan="8">FastCounter[7:0]</td></tr><tr><td>6</td><td colspan="7">Reserved</td><td>FC[8]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0	0	1	1	2	CTC[7:0]								3	CTC[15:8]								4	Reserved							0	5	FastCounter[7:0]								6	Reserved							FC[8]
	7	6	5	4	3	2	1	0																																																																								
0	0	0	0	0	0	0	1	0																																																																								
1	0	1	1	1	0	0	1	1																																																																								
2	CTC[7:0]																																																																															
3	CTC[15:8]																																																																															
4	Reserved							0																																																																								
5	FastCounter[7:0]																																																																															
6	Reserved							FC[8]																																																																								
Dependencies	IA32_RTIT_CTL.MTCEn && IA32_RTIT_CTL.TSCEn && TriggerEn			Generation Scenario		Sent with any TSC packet.																																																																										
Description	The TMA packet serves to provide the information needed to allow the decoder to correlate MTC packets with TSC packets. With this packet, when a MTC packet is encountered, the decoder can determine how many timestamp counter ticks have passed since the last TSC or MTC packet. See Section 36.8.3.2 for details on how to make this calculation.																																																																															
Application	TMA is always sent immediately following a TSC packet, and the payload values are consistent with the TSC payload value. Thus the application of TMA matches that of TSC.																																																																															

36.4.2.14 Cycle Count (CYC) Packet

Table 36-33. Cycle Count Packet Definition

Name	Cycle Count (CYC) Packet

Table 36-33. Cycle Count Packet Definition

Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td colspan="5">Cycle Counter[4:0]</td><td>Exp</td><td>1</td><td>1</td></tr><tr><td>1</td><td colspan="7">Cycle Counter[11:5]</td><td>Exp</td></tr><tr><td>2</td><td colspan="7">Cycle Counter[18:12]</td><td>Exp</td></tr><tr><td>...</td><td colspan="8">... (if Exp = 1 in the previous byte)</td></tr></table>					7	6	5	4	3	2	1	0	0	Cycle Counter[4:0]					Exp	1	1	1	Cycle Counter[11:5]							Exp	2	Cycle Counter[18:12]							Exp	...	... (if Exp = 1 in the previous byte)							
	7	6	5	4	3	2	1	0																																									
0	Cycle Counter[4:0]					Exp	1	1																																									
1	Cycle Counter[11:5]							Exp																																									
2	Cycle Counter[18:12]							Exp																																									
...	... (if Exp = 1 in the previous byte)																																																
Dependencies	IA32_RTIT_CTL.CYCEn&& TriggerEn	Generation Scenario	Can be sent at any time, though a maximum of one CYC packet is sent per core clock cycle. See Section 36.3.6 for CYC-eligible packets.																																														
Description	The Cycle Counter field increments at the same rate as the processor core clock ticks, but with a variable length format (using a trailing EXP bit field) and a range-capped byte length. If the CYC value is less than 32, a 1-byte CYC will be generated, with Exp=0. If the CYC value is between 32 and 4095 inclusive, a 2-byte CYC will be generated, with byte 0 Exp=1 and byte 1 Exp=0. And so on. CYC provides the number of core clocks that have passed since the last CYC packet. CYC can be configured to be sent in every cycle in which an eligible packet is generated, or software can opt to use a threshold to limit the number of CYC packets, at the expense of some precision. These settings are configured using the IA32_RTIT_CTL.CycThresh field (see Section 36.2.8.2). For details on Cycle-Accurate Mode, IPC calculation, etc, see Section 36.3.6. When CycThresh=0, and hence no threshold is in use, then a CYC packet will be generated in any cycle in which any CYC-eligible packet is generated. The CYC packet will precede the other packets generated in the cycle, and provides the precise cycle time of the packets that follow. In addition to these CYC packets generated with other packets, CYC packets can be sent stand-alone. These packets serve simply to update the decoder with the number of cycles passed, and are used to ensure that a wrap of the processor's internal cycle counter doesn't cause cycle information to be lost. These stand-alone CYC packets do not indicate the cycle time of any other packet or operation, and will be followed by another CYC packet before any other CYC-eligible packet is seen. When CycThresh>0, CYC packets are generated only after a minimum number of cycles have passed since the last CYC packet. Once this threshold has passed, the behavior above resumes, where CYC will either be sent in the next cycle that produces other CYC-eligible packets, or could be sent stand-alone. When using CYC thresholds, only the cycle time of the operation (instruction or event) that generates the CYC packet is truly known. Other operations simply have their execution time bounded: they completed at or after the last CYC time, and before the next CYC time.																																																
Application	CYC provides the offset cycle time (since the last CYC packet) for the CYC-eligible packet that follows. If another CYC is encountered before the next CYC-eligible packet, the cycle values should be accumulated and applied to the next CYC-eligible packet. If a CYC packet is generated by a TNT, note that the cycle time provided by the CYC packet applies to the first branch in the TNT packet.																																																

36.4.2.15 VMCS Packet

Table 36-34. VMCS Packet Definition

Name	VMCS Packet

Table 36-34. VMCS Packet Definition

Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>2</td><td colspan="8">VMCS pointer [19:12]</td></tr><tr><td>3</td><td colspan="8">VMCS pointer [27:20]</td></tr><tr><td>4</td><td colspan="8">VMCS pointer [35:28]</td></tr><tr><td>5</td><td colspan="8">VMCS pointer [43:36]</td></tr><tr><td>6</td><td colspan="8">VMCS pointer [51:44]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	0	0	0	2	VMCS pointer [19:12]								3	VMCS pointer [27:20]								4	VMCS pointer [35:28]								5	VMCS pointer [43:36]								6	VMCS pointer [51:44]							
	7	6	5	4	3	2	1	0																																																																								
0	0	0	0	0	0	0	1	0																																																																								
1	1	1	0	0	1	0	0	0																																																																								
2	VMCS pointer [19:12]																																																																															
3	VMCS pointer [27:20]																																																																															
4	VMCS pointer [35:28]																																																																															
5	VMCS pointer [43:36]																																																																															
6	VMCS pointer [51:44]																																																																															
Dependencies	TriggerEn && ContextEn; Also in VMX operation.		Generation Scenario		Generated on successful VMPTRLD, and optionally on PSB+, SMM VM exits, and VM entries that return from SMM (see Section 36-53).																																																																											
Description	The VMCS packet provides a VMCS pointer for a decoder to determine the transition of code contexts: - On a successful VMPTRLD (i.e., a VMPTRLD that doesn't fault, fail, or VM exit), the VMCS packet contains the logical processor's VMCS pointer established by VMPTRLD (for subsequent execution of a VM guest context). - An SMM VM exit loads the logical processor's VMCS pointer with the SMM-transfer VMCS pointer. If the "conceal VMX from PT" VM-exit control is 0 (see Section 36.5.1), a VMCS packet provides this pointer. See Section 36.6 on tracing inside and outside STM. - A VM entry that returns from SMM loads the logical processor's VMCS pointer from a field in the SMM-transfer VMCS. If the "conceal VMX from PT" VM-entry control is 0, a VMCS packet provides this pointer. Whether the VM entry is to VMX root operation or VMX non-root operation is indicated by the PIP.NR bit. A VMCS packet generated before a VMCS pointer has been loaded, or after the VMCS pointer has been cleared will set all 64 bits in the VMCS pointer field. VMCS packets will not be seen on processors with IA32_VMX_MISC[bit 14]=0, as these processors do not allow TraceEn to be set in VMX operation.																																																																															
Application	The purpose of the VMCS packet is to help the decoder uniquely identify changes in the executing software context in situations that CR3 may not be unique. When a VMCS packet is encountered, a decoder should do the following: - If there was a prior unbound FUP (that is, a FUP not preceded by a packet such as MODE.TSX that consumes it, and it hence pairs with a TIP that has not yet been seen), then this VMCS is part of a compound packet event (Section 36.4.1). Find the ending TIP and apply the new VMCS base pointer value to the TIP payload IP. - Otherwise, look for the next VMPTRLD, VMRESUME, or VMLAUNCH in the disassembly, and apply the new VMCS base pointer on the next VM entry. For examples of the packets generated by these flows, see Section 36.7.																																																																															

36.4.2.16 Overflow (OVF) Packet

Table 36-35. OVF Packet Definition

Name	Overflow (OVF) Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	1	1	1	1	0	0	1	1
Dependencies	TriggerEn		Generation Scenario		On resolution of internal buffer overflow				

Table 36-35. OVF Packet Definition

Description	OVF simply indicates to the decoder that an internal buffer overflow occurred, and packets were likely lost. If BranchEN= 1, OVF is followed by a FUP or TIP.PGE which will provide the IP at which packet generation resumes. See Section 36.3.8.
Application	When an OVF packet is encountered, the decoder should skip to the IP given in the subsequent FUP or TIP.PGE. The cycle counter for the CYC packet will be reset at the time the OVF packet is sent. Software should reset its call stack depth on overflow, since no RET compression is allowed across an overflow. Similarly, any IP compression that follows the OVF is guaranteed to use as a reference LastIP the IP payload of an IP packet that preceded the overflow.

36.4.2.17 Packet Stream Boundary (PSB) Packet

Table 36-36. PSB Packet Definition

Name	Packet Stream Boundary (PSB) Packet																																																																																																																																																																	
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>3</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>5</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>6</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>7</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>8</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>9</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>10</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>11</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>12</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>13</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>14</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>15</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>										7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	2	0	0	0	0	0	0	1	0	3	1	0	0	0	0	0	1	0	4	0	0	0	0	0	0	1	0	5	1	0	0	0	0	0	1	0	6	0	0	0	0	0	0	1	0	7	1	0	0	0	0	0	1	0	8	0	0	0	0	0	0	1	0	9	1	0	0	0	0	0	1	0	10	0	0	0	0	0	0	1	0	11	1	0	0	0	0	0	1	0	12	0	0	0	0	0	0	1	0	13	1	0	0	0	0	0	1	0	14	0	0	0	0	0	0	1	0	15	1	0	0	0	0	0	1	0
	7	6	5	4	3	2	1	0																																																																																																																																																										
0	0	0	0	0	0	0	1	0																																																																																																																																																										
1	1	0	0	0	0	0	1	0																																																																																																																																																										
2	0	0	0	0	0	0	1	0																																																																																																																																																										
3	1	0	0	0	0	0	1	0																																																																																																																																																										
4	0	0	0	0	0	0	1	0																																																																																																																																																										
5	1	0	0	0	0	0	1	0																																																																																																																																																										
6	0	0	0	0	0	0	1	0																																																																																																																																																										
7	1	0	0	0	0	0	1	0																																																																																																																																																										
8	0	0	0	0	0	0	1	0																																																																																																																																																										
9	1	0	0	0	0	0	1	0																																																																																																																																																										
10	0	0	0	0	0	0	1	0																																																																																																																																																										
11	1	0	0	0	0	0	1	0																																																																																																																																																										
12	0	0	0	0	0	0	1	0																																																																																																																																																										
13	1	0	0	0	0	0	1	0																																																																																																																																																										
14	0	0	0	0	0	0	1	0																																																																																																																																																										
15	1	0	0	0	0	0	1	0																																																																																																																																																										
Dependencies	TriggerEn	Generation Scenario	Periodic, based on the number of output bytes generated while tracing. PSB is sent when IA32_RTIT_STATUS.PacketByteCnt=0, and each time it crosses the software selected threshold after that. May be sent for other micro-architectural conditions as well.																																																																																																																																																															
Description	PSB is a unique pattern in the packet output log, and hence serves as a sync point for the decoder. It is a pattern that the decoder can search for in order to get aligned on packet boundaries. This packet is periodic, based on the number of output bytes, as indicated by IA32_RTIT_STATUS.PacketByteCnt. The period is chosen by software, via IA32_RTIT_CTL.PSBFreq (see Section 36.2.8.2). Note, however, that the PSB period is not precise, it simply reflects the average number of output bytes that should pass between PSBs. The processor will make a best effort to insert PSB as quickly after the selected threshold is reached as possible. The processor also may send extra PSB packets for some micro-architectural conditions. PSB also serves as the leading packet for a set of “status-only” packets collectively known as PSB+ (Section 36.3.7).																																																																																																																																																																	

Table 36-36. PSB Packet Definition (Contd.)

Application	When a PSB is seen, the decoder should interpret all following packets as “status only”, until either a PSBEND or OVF packet is encountered. “Status only” implies that the binding and ordering rules to which these packets normally adhere are ignored, and the state they carry can instead be applied to the IP payload in the FUP packet that is included.

36.4.2.18 PSBEND Packet**Table 36-37. PSBEND Packet Definition**

Name	PSBEND Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	0	0	1	0	0	0	1	1
Dependencies	TriggerEn			Generation Scenario	Always follows PSB packet, separated by PSB+ packets				
Description	PSBEND is simply a terminator for the series of “status only” (PSB+) packets that follow PSB (Section 36.3.7).								
Application	When a PSBEND packet is seen, the decoder should cease to treat packets as “status only”.								

36.4.2.19 Maintenance (MNT) Packet**Table 36-38. MNT Packet Definition**

Name	Maintenance (MNT) Packet								
Packet Format		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	1	1	0	0	0	0	1	1
	2	1	0	0	0	1	0	0	0
	3	Payload[7:0]							
	4	Payload[15:8]							
	5	Payload[23:16]							
	6	Payload[31:24]							
	7	Payload[39:32]							
	8	Payload[47:40]							
	9	Payload[55:48]							
	10	Payload[63:56]							
Dependencies	TriggerEn			Generation Scenario		Implementation specific.			
Description	This packet is generated by hardware, the payload meaning is model-specific.								
Application	Unless a decoder has been extended for a particular family/model/stepping to interpret MNT packet payloads, this packet should simply be ignored. It does not bind to any IP.								

36.4.2.20 PAD Packet

Table 36-39. PAD Packet Definition

Name	PAD Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	0
Dependencies	TriggerEn			Generation Scenario	Implementation specific				
Description	PAD is simply a NOP packet. Processor implementations may choose to add pad packets to improve packet alignment or for implementation-specific reasons.								
Application	Ignore PAD packets.								

36.4.2.21 PTWRITE (PTW) Packet

Table 36-40. PTW Packet Definition

Name	PTW Packet																																																																																																										
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>IP</td><td colspan="2">PayloadBytes</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>2</td><td colspan="8">Payload[7:0]</td></tr><tr><td>3</td><td colspan="8">Payload[15:8]</td></tr><tr><td>4</td><td colspan="8">Payload[23:16]</td></tr><tr><td>5</td><td colspan="8">Payload[31:24]</td></tr><tr><td>6</td><td colspan="8">Payload[39:32]</td></tr><tr><td>7</td><td colspan="8">Payload[47:40]</td></tr><tr><td>8</td><td colspan="8">Payload[55:48]</td></tr><tr><td>9</td><td colspan="8">Payload[63:56]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	IP	PayloadBytes		1	0	0	1	0	2	Payload[7:0]								3	Payload[15:8]								4	Payload[23:16]								5	Payload[31:24]								6	Payload[39:32]								7	Payload[47:40]								8	Payload[55:48]								9	Payload[63:56]							
	7	6	5	4	3	2	1	0																																																																																																			
0	0	0	0	0	0	0	1	0																																																																																																			
1	IP	PayloadBytes		1	0	0	1	0																																																																																																			
2	Payload[7:0]																																																																																																										
3	Payload[15:8]																																																																																																										
4	Payload[23:16]																																																																																																										
5	Payload[31:24]																																																																																																										
6	Payload[39:32]																																																																																																										
7	Payload[47:40]																																																																																																										
8	Payload[55:48]																																																																																																										
9	Payload[63:56]																																																																																																										
	The PayloadBytes field indicates the number of bytes of payload that follow the header bytes. Encodings are as follows: <table><tr><th>PayloadBytes</th><th>Bytes of Payload</th></tr><tr><td>'00</td><td>4</td></tr><tr><td>'01</td><td>8</td></tr><tr><td>'10</td><td>Reserved</td></tr><tr><td>'11</td><td>Reserved</td></tr></table> IP bit indicates if a FUP, whose payload will be the IP of the PTWRITE instruction, will follow.								PayloadBytes	Bytes of Payload	'00	4	'01	8	'10	Reserved	'11	Reserved																																																																																									
PayloadBytes	Bytes of Payload																																																																																																										
'00	4																																																																																																										
'01	8																																																																																																										
'10	Reserved																																																																																																										
'11	Reserved																																																																																																										
Dependencies	TriggerEn && ContextEn && FilterEn && PTWEn		Generation Scenario		PTWRITE Instruction																																																																																																						

Table 36-40. PTW Packet Definition (Contd.)

Description	Contains the value held in the PTWRITE operand. This packet is CYC-eligible, and hence will generate a CYC packet if IA32_RTIT_CTL.CYCEn=1 and any CYC Threshold has been reached.
Application	Binds to the associated PTWRITE instruction. The IP of the PTWRITE will be provided by a following FUP, when PTW.IP=1.

36.4.2.22 Execution Stop (EXSTOP) Packet

Table 36-41. EXSTOP Packet Definition

Name	EXSTOP Packet																													
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>IP</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table> IP bit indicates if a FUP will follow.				7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	IP	1	1	0	0	0	1	0
	7	6	5	4	3	2	1	0																						
0	0	0	0	0	0	0	1	0																						
1	IP	1	1	0	0	0	1	0																						
Dependencies	TriggerEn && PwrEvtEn	Generation Scenario	C-state entry, P-state change, or other processor clock power-down. Includes : ▪ Entry to C-state deeper than C0.0 ▪ TM1/2 ▪ STPCLK# ▪ Frequency change due to IA32_CLOCK_MODULATION, Turbo																											
Description	This packet indicates that software execution has stopped due to processor clock powerdown. Later packets will indicate when execution resumes. If EXSTOP is generated while ContextEn is set, the IP bit will be set, and EXSTOP will be followed by a FUP packet containing the IP at which execution stopped. More precisely, this will be the IP of the oldest instruction that has not yet completed. This packet is CYC-eligible, and hence will generate a CYC packet if IA32_RTIT_CTL.CYCEn=1 and any CYC Threshold has been reached.																													
Application	If a FUP follows EXSTOP (hence IP bit set), the EXSTOP can be bound to the FUP IP. Otherwise the IP is not known. Time of powerdown can be inferred from the preceding CYC, if CYCEn=1. Combined with the TSC at the time of wake (if TSCEn=1), this can be used to determine the duration of the powerdown.																													

36.4.2.23 MWAIT Packet

Table 36-42. MWAIT Packet Definition

Name	MWAIT Packet

Table 36-42. MWAIT Packet Definition (Contd.)

Packet Format		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	1	1	0	0	0	0	1	0
	2	MWAIT Hints[7:0]							
	3	Reserved							
	4	Reserved							
	5	Reserved							
	6	Reserved						EXT[1:0]	
	7	Reserved							
	8	Reserved							
	9	Reserved							
Dependencies	TriggerEn && PwrEvtEn && ContextEn			Generation Scenario	MWAIT, UMWAIT, or TPAUSE instructions, or I/O redirection to MWAIT, that complete without fault or VMexit.				
Description	Indicates that an MWAIT operation to C-state deeper than C0.0 completed. The MWAIT hints and extensions passed in by software are exposed in the payload. For UMWAIT and TPAUSE, the EXT field holds the input register value that determines the optimized state requested. For entry to some highly optimized C0 sub-C-states, such as C0.1, no MWAIT packet is generated. This packet is CYC-eligible, and hence will generate a CYC packet if IA32_RTIT_CTL.CYCEn=1 and any CYC Threshold has been reached.								
Application	The binding for the upcoming EXSTOP packet also applies to the MWAIT packet. See Section 36.4.2.22.								

36.4.2.24 Power Entry (PWRE) Packet

Table 36-43. PWRE Packet Definition

Name	PWRE Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	0	0	1	0	0	0	1	0
	2	HW	Reserved						
	3	Resolved Thread C-State				Resolved Thread Sub C-State			
Dependencies	TriggerEn && PwrEvtEn			Generation Scenario	Transition to a C-state deeper than C0.0.				

Table 36-43. PWRE Packet Definition (Contd.)

Description	Indicates processor entry to the resolved thread C-state and sub C-state indicated. The processor will remain in this C-state until either another PWRE indicates the processor has moved to a C-state deeper than C0.0, or a PWRX packet indicates a return to C0.0. For entry to some highly optimized C0 sub-C-states, such as C0.1, no PWRE packet is generated. Note that some CPUs may allow MWAIT to request a deeper C-state than is supported by the core. These deeper C-states may have platform-level implications that differentiate them. However, the PWRE packet will provide only the resolved thread C-state, which will not exceed that supported by the core. If the C-state entry was initiated by hardware, rather than a direct software request (such as MWAIT, UMWAIT, TPAUSE, HLT, or shutdown), the HW bit will be set to indicate this. Hardware Duty Cycling (see Section 17.5, “Hardware Duty Cycling (HDC),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B) is an example of such a case.
Application	When transitioning from C0.0 to a deeper C-state, the PWRE packet will be followed by an EXSTOP. If that EXSTOP packet has the IP bit set, then the following FUP will provide the IP at which the C-state entry occurred. Subsequent PWRE packets generated before the next PWRX should bind to the same IP.

36.4.2.25 Power Exit (PWRX) Packet

Table 36-44. PWRX Packet Definition

Name	PWRX Packet																																																																															
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>2</td><td colspan="4">Last Core C-State</td><td colspan="4">Deepest Core C-State</td></tr><tr><td>3</td><td colspan="4">Reserved</td><td colspan="4">Wake Reason</td></tr><tr><td>4</td><td colspan="8">Reserved</td></tr><tr><td>5</td><td colspan="8">Reserved</td></tr><tr><td>6</td><td colspan="8">Reserved</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	1	0	2	Last Core C-State				Deepest Core C-State				3	Reserved				Wake Reason				4	Reserved								5	Reserved								6	Reserved							
	7	6	5	4	3	2	1	0																																																																								
0	0	0	0	0	0	0	1	0																																																																								
1	1	0	1	0	0	0	1	0																																																																								
2	Last Core C-State				Deepest Core C-State																																																																											
3	Reserved				Wake Reason																																																																											
4	Reserved																																																																															
5	Reserved																																																																															
6	Reserved																																																																															
Dependencies	TriggerEn && PwrEvtEn			Generation Scenario	Transition from a C-state deeper than C0.0 to C0.																																																																											
Description	Indicates processor return to thread C0 from a C-state deeper than C0.0. For return from some highly optimized C0 sub-C-states, such as C0.1, no PWRX packet is generated. The Last Core C-State field provides the MWAIT encoding for the core C-state at the time of the wake. The Deepest Core C-State provides the MWAIT encoding for the deepest core C-state achieved during the sleep session, or since leaving thread C0. MWAIT encodings for C-states can be found in Table 4-11 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B. Note that these values reflect only the core C-state, and hence will not exceed the maximum supported core C-state, even if deeper C-states can be requested. The Wake Reason field is one-hot, encoded as follows: <table><tr><td>Bit</td><td>Field</td><td>Meaning</td></tr><tr><td>0</td><td>Interrupt</td><td>Wake due to external interrupt received.</td></tr><tr><td>1</td><td>Timer Deadline</td><td>Wake due to timer expiration, such as UMWAIT/TPAUSE TSC-quanta.</td></tr><tr><td>2</td><td>Store to Monitored Address</td><td>Wake due to store to monitored address.</td></tr><tr><td>3</td><td>HW Wake</td><td>Wake due to hardware autonomous condition, such as HDC.</td></tr></table>								Bit	Field	Meaning	0	Interrupt	Wake due to external interrupt received.	1	Timer Deadline	Wake due to timer expiration, such as UMWAIT/TPAUSE TSC-quanta.	2	Store to Monitored Address	Wake due to store to monitored address.	3	HW Wake	Wake due to hardware autonomous condition, such as HDC.																																																									
Bit	Field	Meaning																																																																														
0	Interrupt	Wake due to external interrupt received.																																																																														
1	Timer Deadline	Wake due to timer expiration, such as UMWAIT/TPAUSE TSC-quanta.																																																																														
2	Store to Monitored Address	Wake due to store to monitored address.																																																																														
3	HW Wake	Wake due to hardware autonomous condition, such as HDC.																																																																														

Table 36-44. PWRX Packet Definition (Contd.)

Application	PWRX will always apply to the same IP as the PWRE. The time of wake can be discerned from (optional) timing packets that precede PWRX.

36.4.2.26 Block Begin Packet (BBP)**Table 36-45. Block Begin Packet Definition**

Name	BBP																																											
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>2</td><td>SZ</td><td colspan="2">Reserved</td><td colspan="5">Type[4:0]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0	1	1	2	SZ	Reserved		Type[4:0]				
	7	6	5	4	3	2	1	0																																				
0	0	0	0	0	0	0	1	0																																				
1	0	1	1	0	0	0	1	1																																				
2	SZ	Reserved		Type[4:0]																																								
Dependencies	TriggerEn		Generation Scenario		PEBS event, if IA32_PEBS_ENABLE.OUTPUT=1.																																							
Description	This packet indicates the beginning of a block of packets which are collectively tied to a single event or instruction. The size of the block item payloads within this block is provided by the Size (SZ) bit: SZ=0: 8-byte block items SZ=1: 4-byte block items The meaning of the BIP payloads is provided by the Type field: <table><tr><th>BBP.Type</th><th>Block name</th></tr><tr><td>0x00</td><td>Reserved</td></tr><tr><td>0x01</td><td>General-Purpose Registers</td></tr><tr><td>0x02..0x03</td><td>Reserved</td></tr><tr><td>0x04</td><td>PEBS Basic</td></tr><tr><td>0x05</td><td>PEBS Memory</td></tr><tr><td>0x06..0x07</td><td>Reserved</td></tr><tr><td>0x08</td><td>LBR Block 0</td></tr><tr><td>0x09</td><td>LBR Block 1</td></tr><tr><td>0x0A</td><td>LBR Block 2</td></tr><tr><td>0x0B..0x0F</td><td>Reserved</td></tr><tr><td>0x10</td><td>XMM Registers</td></tr><tr><td>0x11..0x1F</td><td>Reserved</td></tr></table>								BBP.Type	Block name	0x00	Reserved	0x01	General-Purpose Registers	0x02..0x03	Reserved	0x04	PEBS Basic	0x05	PEBS Memory	0x06..0x07	Reserved	0x08	LBR Block 0	0x09	LBR Block 1	0x0A	LBR Block 2	0x0B..0x0F	Reserved	0x10	XMM Registers	0x11..0x1F	Reserved										
BBP.Type	Block name																																											
0x00	Reserved																																											
0x01	General-Purpose Registers																																											
0x02..0x03	Reserved																																											
0x04	PEBS Basic																																											
0x05	PEBS Memory																																											
0x06..0x07	Reserved																																											
0x08	LBR Block 0																																											
0x09	LBR Block 1																																											
0x0A	LBR Block 2																																											
0x0B..0x0F	Reserved																																											
0x10	XMM Registers																																											
0x11..0x1F	Reserved																																											
Application	A BBP will always be followed by a Block End Packet (BEP), and when the block is generated while ContextEn=1 that BEP will have IP=1 and be followed by a FUP that provides the IP to which the block should be bound. Note that, in addition to BEP, a block can be terminated by a BBP (indicating the start of a new block) or an OVF packet.																																											

36.4.2.27 Block Item Packet (BIP)**Table 36-46. Block Item Packet Definition**

Name	BIP

Table 36-46. Block Item Packet Definition (Contd.)

Packet Format	If the preceding BBP.SZ=0:								
		7	6	5	4	3	2	1	0
	0	ID[5:0]					1	0	0
	1	Payload[7:0]							
	2	Payload[15:8]							
	3	Payload[23:16]							
	4	Payload[31:24]							
	5	Payload[39:32]							
	6	Payload[47:40]							
	7	Payload[55:48]							
	8	Payload[63:56]							
	If the preceding BBP.SZ=1:								
		7	6	5	4	3	2	1	0
	0	ID[5:0]					1	0	0
	1	Payload[7:0]							
2	Payload[15:8]								
3	Payload[23:16]								
4	Payload[31:24]								
Dependencies	TriggerEn			Generation Scenario		See BBP.			
Description	The size of the BIP payload is determined by the Size field in the preceding BBP packet. The BIP header provides the ID value that, when combined with the Type field from the preceding BBP, uniquely identifies the state value held in the BIP payload. See Table 36-47 below for the complete list.								
Application	See BBP.								

BIP State Value Encodings

The table below provides the encoding values for all defined block items. State items that are larger than 8 bytes, such as XMM register values, are broken into multiple 8-byte components. BIP packets with Size=1 (4 byte payload) will provide only the lower 4 bytes of the associated state value.

Table 36-47. BIP Encodings

BBP.Type	BIP.ID	State Value
General-Purpose Registers		
0x01	0x00	R/EFLAGS
0x01	0x01	R/EIP
0x01	0x02	R/EAX
0x01	0x03	R/ECX
0x01	0x04	R/EDX
0x01	0x05	R/EBX
0x01	0x06	R/ESP

Table 36-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x01	0x07	R/EBP
0x01	0x08	R/ESI
0x01	0x09	R/EDI
0x01	0x0A	R8
0x01	0x0B	R9
0x01	0x0C	R10
0x01	0x0D	R11
0x01	0x0E	R12
0x01	0x0F	R13
0x01	0x10	R14
0x01	0x11	R15
PEBS Basic Info (Section 22.9.2.2.1)		
0x04	0x00	Instruction Pointer
0x04	0x01	Applicable Counters
0x04	0x02	Timestamp
PEBS Memory Info (Section 22.9.2.2.2)		
0x05	0x00	MemAccessAddress
0x05	0x01	MemAuxInfo
0x05	0x02	MemAccessLatency
0x05	0x03	TSXAuxInfo
LBR_0		
0x08	0x00	LBR[TOS-0]_FROM_IP
0x08	0x01	LBR[TOS-0]_TO_IP
0x08	0x02	LBR[TOS-0]_INFO
0x08	0x03	LBR[TOS-1]_FROM_IP
0x08	0x04	LBR[TOS-1]_TO_IP
0x08	0x05	LBR[TOS-1]_INFO
0x08	0x06	LBR[TOS-2]_FROM_IP
0x08	0x07	LBR[TOS-2]_TO_IP
0x08	0x08	LBR[TOS-2]_INFO
0x08	0x09	LBR[TOS-3]_FROM_IP
0x08	0x0A	LBR[TOS-3]_TO_IP
0x08	0x0B	LBR[TOS-3]_INFO
0x08	0x0C	LBR[TOS-4]_FROM_IP
0x08	0x0D	LBR[TOS-4]_TO_IP
0x08	0x0E	LBR[TOS-4]_INFO
0x08	0x0F	LBR[TOS-5]_FROM_IP
0x08	0x10	LBR[TOS-5]_TO_IP

Table 36-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x08	0x11	LBR[TOS-5]_INFO
0x08	0x12	LBR[TOS-6]_FROM_IP
0x08	0x13	LBR[TOS-6]_TO_IP
0x08	0x14	LBR[TOS-6]_INFO
0x08	0x15	LBR[TOS-7]_FROM_IP
0x08	0x16	LBR[TOS-7]_TO_IP
0x08	0x17	LBR[TOS-7]_INFO
0x08	0x18	LBR[TOS-8]_FROM_IP
0x08	0x19	LBR[TOS-8]_TO_IP
0x08	0x1A	LBR[TOS-8]_INFO
0x08	0x1B	LBR[TOS-9]_FROM_IP
0x08	0x1C	LBR[TOS-9]_TO_IP
0x08	0x1D	LBR[TOS-9]_INFO
0x08	0x1E	LBR[TOS-10]_FROM_IP
0x08	0x1F	LBR[TOS-10]_TO_IP
LBR_1		
0x09	0x00	LBR[TOS-10]_INFO
0x09	0x01	LBR[TOS-11]_FROM_IP
0x09	0x02	LBR[TOS-11]_TO_IP
0x09	0x03	LBR[TOS-11]_INFO
0x09	0x04	LBR[TOS-12]_FROM_IP
0x09	0x05	LBR[TOS-12]_TO_IP
0x09	0x06	LBR[TOS-12]_INFO
0x09	0x07	LBR[TOS-13]_FROM_IP
0x09	0x08	LBR[TOS-13]_TO_IP
0x09	0x09	LBR[TOS-13]_INFO
0x09	0x0A	LBR[TOS-14]_FROM_IP
0x09	0x0B	LBR[TOS-14]_TO_IP
0x09	0x0C	LBR[TOS-14]_INFO
0x09	0x0D	LBR[TOS-15]_FROM_IP
0x09	0x0E	LBR[TOS-15]_TO_IP
0x09	0x0F	LBR[TOS-15]_INFO
0x09	0x10	LBR[TOS-16]_FROM_IP
0x09	0x11	LBR[TOS-16]_TO_IP
0x09	0x12	LBR[TOS-16]_INFO
0x09	0x13	LBR[TOS-17]_FROM_IP
0x09	0x14	LBR[TOS-17]_TO_IP
0x09	0x15	LBR[TOS-17]_INFO

Table 36-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x09	0x16	LBR[TOS-18]_FROM_IP
0x09	0x17	LBR[TOS-18]_TO_IP
0x09	0x18	LBR[TOS-18]_INFO
0x09	0x19	LBR[TOS-19]_FROM_IP
0x09	0x1A	LBR[TOS-19]_TO_IP
0x09	0x1B	LBR[TOS-19]_INFO
0x09	0x1C	LBR[TOS-20]_FROM_IP
0x09	0x1D	LBR[TOS-20]_TO_IP
0x09	0x1E	LBR[TOS-20]_INFO
0x09	0x1F	LBR[TOS-21]_FROM_IP
LBR_2		
0x0A	0x00	LBR[TOS-21]_TO_IP
0x0A	0x01	LBR[TOS-21]_INFO
0x0A	0x02	LBR[TOS-22]_FROM_IP
0x0A	0x03	LBR[TOS-22]_TO_IP
0x0A	0x04	LBR[TOS-22]_INFO
0x0A	0x05	LBR[TOS-23]_FROM_IP
0x0A	0x06	LBR[TOS-23]_TO_IP
0x0A	0x07	LBR[TOS-23]_INFO
0x0A	0x08	LBR[TOS-24]_FROM_IP
0x0A	0x09	LBR[TOS-24]_TO_IP
0x0A	0x0A	LBR[TOS-24]_INFO
0x0A	0x0B	LBR[TOS-25]_FROM_IP
0x0A	0x0C	LBR[TOS-25]_TO_IP
0x0A	0x0D	LBR[TOS-25]_INFO
0x0A	0x0E	LBR[TOS-26]_FROM_IP
0x0A	0x0F	LBR[TOS-26]_TO_IP
0x0A	0x10	LBR[TOS-26]_INFO
0x0A	0x11	LBR[TOS-27]_FROM_IP
0x0A	0x12	LBR[TOS-27]_TO_IP
0x0A	0x13	LBR[TOS-27]_INFO
0x0A	0x14	LBR[TOS-28]_FROM_IP
0x0A	0x15	LBR[TOS-28]_TO_IP
0x0A	0x16	LBR[TOS-28]_INFO
0x0A	0x17	LBR[TOS-29]_FROM_IP
0x0A	0x18	LBR[TOS-29]_TO_IP
0x0A	0x19	LBR[TOS-29]_INFO
0x0A	0x1A	LBR[TOS-30]_FROM_IP

Table 36-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x0A	0x1B	LBR[TOS-30]_TO_IP
0x0A	0x1C	LBR[TOS-30]_INFO
0x0A	0x1D	LBR[TOS-31]_FROM_IP
0x0A	0x1E	LBR[TOS-31]_TO_IP
0x0A	0x1F	LBR[TOS-31]_INFO
XMM Registers		
0x10	0x00	XMM0_Q0
0x10	0x01	XMM0_Q1
0x10	0x02	XMM1_Q0
0x10	0x03	XMM1_Q1
0x10	0x04	XMM2_Q0
0x10	0x05	XMM2_Q1
0x10	0x06	XMM3_Q0
0x10	0x07	XMM3_Q1
0x10	0x08	XMM4_Q0
0x10	0x09	XMM4_Q1
0x10	0x0A	XMM5_Q0
0x10	0x0B	XMM5_Q1
0x10	0x0C	XMM6_Q0
0x10	0x0D	XMM6_Q1
0x10	0x0E	XMM7_Q0
0x10	0x0F	XMM7_Q1
0x10	0x10	XMM8_Q0
0x10	0x11	XMM8_Q1
0x10	0x12	XMM9_Q0
0x10	0x13	XMM9_Q1
0x10	0x14	XMM10_Q0
0x10	0x15	XMM10_Q1
0x10	0x16	XMM11_Q0
0x10	0x17	XMM11_Q1
0x10	0x18	XMM12_Q0
0x10	0x19	XMM12_Q1
0x10	0x1A	XMM13_Q0
0x10	0x1B	XMM13_Q1
0x10	0x1C	XMM14_Q0
0x10	0x1D	XMM14_Q1
0x10	0x1E	XMM15_Q0
0x10	0x1F	XMM15_Q1

36.4.2.28 Block End Packet (BEP)

Table 36-48. Block End Packet Definition

Name	BEP								
Packet Format									
		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	IP	0	1	1	0	0	1	1
Dependencies	TriggerEn			Generation Scenario		See BBP.			
Description	Indicates the end of a packet block. The IP bit indicates if a FUP will follow, and will be set if ContextEn=1.								
Application	The block, from initial BBP to the BEP, binds to the FUP IP, if IP=1, and consumes the FUP.								

36.4.2.29 Control Flow Event (CFE) Packet

Table 36-49. Control Flow Event Packet Definition

Name	CFE																																															
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>2</td><td>IP</td><td colspan="2">Reserved</td><td colspan="5">Type[4:0]</td></tr><tr><td>3</td><td colspan="8">Vector[7:0]</td></tr></table> IP bit indicates if a stand-alone FUP will follow.				7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	2	IP	Reserved		Type[4:0]					3	Vector[7:0]							
	7	6	5	4	3	2	1	0																																								
0	0	0	0	0	0	0	1	0																																								
1	0	0	0	1	0	0	1	1																																								
2	IP	Reserved		Type[4:0]																																												
3	Vector[7:0]																																															
Dependencies	IA32_RTIT_CTL.EventEn && TriggerEn && ContextEn On ContextEn transitions, the CFE will be generated regardless of direction (1→0 or 0→1). VM exit is an exception, where CFE.VMEXIT depends only on the prior value of ContextEn.	Generation Scenario	Software interrupt (including SYSCALL and SYSENTER when FRED transitions are enabled), external interrupt, user interrupt, or exception, including those injected on VM entry. INIT, SIPI, SMI, RSM, ERETS, ERETU, IRET, Shutdown. VM exit, if “Conceal VMX in PT” VMCS exit control is 0. VM entry, if “Conceal VMX in PT” VMCS entry control is 0. TSX Abort.																																													
Description	This packet indicates that an asynchronous event or related event (see list above) has occurred. The type of event is provided in the packet (see Table 36-50 below), and, if the IP bit is set, the IP at which the event occurred is provided in a stand-alone FUP packet that follows. Further, in the case of an interrupt or exception, the vector field provides the vector of the event. The IP bit will be set only when ContextEn=1 before the event is taken, and either BranchEn=0 or else no FUP is generated for this event by BranchEn=1. There are some cases, such as SIPI and RSM, where no FUP is generated. Note that events that are not delivered to software, such as nested events or events which cause a VM exit, do not generate CFE packets.																																															
Application	If the IP bit is set, a FUP will follow that is stand-alone (not part of a compound packet event), and the CFE consumes the FUP. If the IP bit is not set, the CFE binds to the next FUP if PacketEn=1 (hence the CFE comes after a TIP.PGE but before the next TIP.PGD), and is stand-alone if PacketEn=0.																																															

CFE Packet Type and Vector Fields

Every CFE has a Type field, which provides the type of event which generated the packet. For a subset of CFE Types, the CFE.Vector field may be valid. Details on these fields, as well as the IP to be expected in any following FUP packet, are provided in the table below.

Table 36-50. CFE Packet Type and Vector Fields Details

CFE Subtype	Type	Vector	FUP IP	Details
INTR	0x1	Event Vector	Varies	Used for interrupts (external and software), exceptions, faults, and NMI. FUP contains the address of the instruction that has not completed (NLIP for trap events, CLIP for fault events). <i>Not used for INT n, INT3, INTO, or INT1 when FRED transitions are enabled.</i>
IRET	0x2	Invalid	CLIP	<i>Used by ERETS and ERETU as well as IRET.</i>
SMI	0x3	Invalid	NLIP	
RSM	0x4	Invalid	None	
SIPI	0x5	SIPI Vector	None	
INIT	0x6	Invalid	NLIP	
VMENTRY	0x7	Invalid	CLIP	FUP contains IP of VMLAUNCH/VMRESUME.
VMEXIT	0x8	Invalid	Varies	FUP IP varies depending on type of VM exit, but will be the address of the instruction that has not completed. Will be consistent with Guest IP saved in VMCS.
VMEXIT_INTR	0x9	Event Vector	Varies	Sent in cases where VM exit was caused by an INTR event (interrupt, exception, fault, or NMI). Vector provided is for the event which caused the VM exit. FUP IP behavior matches that of INTR type above.
SHUTDOWN	0xa	Invalid	Varies	FUP IP varies depending on the type of event that caused shutdown, but will be the address of the instruction that has not completed.
Reserved	0xb	N/A	N/A	
UINTR	0xc	User Interrupt Vector	NLIP	User interrupt delivered.
UIRET	0xd	Invalid	CLIP	Exiting from user interrupt routine.
SWINTR	0xe	Event Vector	CLIP	Used for software interrupts (INT n , INT3, INTO, and INT1) when FRED transitions are enabled.
SYSCALL	0xf	Invalid	CLIP	Used for SYSCALL and SYSENTER, but only when FRED transitions are enabled.
Reserved	0x10–0x1f	N/A	N/A	Reserved

36.4.2.30 Event Data (EVD) Packet

Table 36-51. Event Data Packet Definition

Name	EVD

Table 36-51. Event Data Packet Definition (Contd.)

Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>2</td><td colspan="2">Reserved</td><td colspan="6">Type[5:0]</td></tr><tr><td>3</td><td colspan="8">Payload[7:0]</td></tr><tr><td>4</td><td colspan="8">Payload[15:8]</td></tr><tr><td>5</td><td colspan="8">Payload[23:16]</td></tr><tr><td>6</td><td colspan="8">Payload[31:24]</td></tr><tr><td>7</td><td colspan="8">Payload[39:32]</td></tr><tr><td>8</td><td colspan="8">Payload[47:40]</td></tr><tr><td>9</td><td colspan="8">Payload[55:48]</td></tr><tr><td>10</td><td colspan="8">Payload[63:56]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	1	1	2	Reserved		Type[5:0]						3	Payload[7:0]								4	Payload[15:8]								5	Payload[23:16]								6	Payload[31:24]								7	Payload[39:32]								8	Payload[47:40]								9	Payload[55:48]								10	Payload[63:56]							
	7	6	5	4	3	2	1	0																																																																																																												
0	0	0	0	0	0	0	1	0																																																																																																												
1	0	1	0	1	0	0	1	1																																																																																																												
2	Reserved		Type[5:0]																																																																																																																	
3	Payload[7:0]																																																																																																																			
4	Payload[15:8]																																																																																																																			
5	Payload[23:16]																																																																																																																			
6	Payload[31:24]																																																																																																																			
7	Payload[39:32]																																																																																																																			
8	Payload[47:40]																																																																																																																			
9	Payload[55:48]																																																																																																																			
10	Payload[63:56]																																																																																																																			
Dependencies	IA32_RTIT_CTL.EventEn && TriggerEn && ContextEn		Generation Scenario	Page fault, including those injected on VM entry. VM exit, if “Suppress VMX packets on exit” VMCS exit control is 0.																																																																																																																
Description	Provides additional data about the event that caused the following CFE. The Payload field is dictated by the Type. <table><tr><th>Type</th><th>Payload</th></tr><tr><td>'000000</td><td>Page Fault Linear Address, same as CR2 (PFA)</td></tr><tr><td>'000001</td><td>VMX Exit Qualification (VMXQ)</td></tr><tr><td>'000010</td><td>VMX Exit Reason (VMXR)</td></tr><tr><td>'000011 - '111111</td><td>Reserved</td></tr></table> EVD packets are never generated in cases where a CFE is not.								Type	Payload	'000000	Page Fault Linear Address, same as CR2 (PFA)	'000001	VMX Exit Qualification (VMXQ)	'000010	VMX Exit Reason (VMXR)	'000011 - '111111	Reserved																																																																																																		
Type	Payload																																																																																																																			
'000000	Page Fault Linear Address, same as CR2 (PFA)																																																																																																																			
'000001	VMX Exit Qualification (VMXQ)																																																																																																																			
'000010	VMX Exit Reason (VMXR)																																																																																																																			
'000011 - '111111	Reserved																																																																																																																			
Application	EVD packets bind to the same IP (if any) as the subsequent CFE packet.																																																																																																																			

36.5 TRACING IN VMX OPERATION

On processors that IA32_VMX_MISC[bit 14] reports 1, TraceEn can be set in VMX operation. The VMM can configure specific VMX controls to control what virtualization-specific data is included within the trace packets (see Section 36.5.1 for details). The VMM can also configure the VMCS to limit tracing to non-root operation, or to trace across both root and non-root operation. The VMCS controls exist to simplify virtualization of Intel PT for guest use, including the “Clear IA32_RTIT_CTL” exit control (See Section 27.7.1), “Load IA32_RTIT_CTL” entry control (See Section 27.8.1), and “Intel PT uses guest physical addresses” execution control (See Section 28.5.3).

For older processors that do not support these VMCS controls, the MSR-load areas used by VMX transitions can be employed by the VMM to restrict tracing to the desired context. See Section 36.5.2 for details. Tracing with SMM Transfer Monitor is described in Section 36.6.

36.5.1 VMX-Specific Packets and VMCS Controls

In all of the usages of VMX and Intel PT, a decoder in the host or VMM context can identify the occurrences of VMX transitions with the aid of VMX-specific packets. There are four kinds of packets relevant to VMX:

- **VMCS packet.** The VMX transitions of individual VMs can be distinguished by a decoder using the VMCS-pointer field in a VMCS packet. A VMCS packet is sent on a successful execution of VMPTRLD, and its VMCS-pointer field stores the VMCS pointer loaded by that execution. See Section 36.4.2.15 for details.

- **The NR (non-root) bit in a PIP packet.** Normally, the NR bit is set in any PIP packet generated in VMX non-root operation. In addition, PIP packets are generated with each VM entry and VM exit. Thus a transition of the NR bit from 0 to 1 indicates the occurrence of a VM entry, and a transition of 1 to 0 indicates the occurrence of a VM exit.
- **CFE packet.** Identifies VM exit and VM entry operations.
- **EVD packet.** Provides the exit reason and exit qualification for VM exits.

There are VMX controls that a VMM can set to conceal some of this VMX-specific information (by suppressing its recording) and thereby prevent it from leaking across virtualization boundaries. There is one of these controls (each of which is called “conceal VMX from PT”) of each type of VMX control.

Table 36-52. VMX Controls For Intel Processor Trace

Type of VMX Control	Bit Position ¹	Value	Behavior
Secondary processor-based VM-execution control	19	0	Each PIP generated in VM non-root operation will set the NR bit. PSB+ in VMX non-root operation will include the VMCS packet, to ensure that the decoder knows which guest is currently in use.
		1	Each PIP generated in VMX non-root operation will clear the NR bit. PSB+ in VMX non-root operation will not include the VMCS packet.
VM-exit control	24	0	Each VM exit generates a PIP in which the NR bit is clear, and a CFE/EVD if Event Trace is enabled. In addition, SMM VM exits generate VMCS packets.
		1	VM exits do not generate PIPs, CFEs, or EVDs, and no VMCS packets are generated on SMM VM exits.
VM-entry control	17	0	Each VM entry generates a PIP in which the NR bit is set (except VM entries that return from SMM to VMX root operation), and a CFE if Event Trace is enabled. In addition, VM entries that return from SMM generate VMCS packets.
		1	VM entries do not generate PIPs or CFEs, and no VMCS packets are generated on VM entries that return from SMM.

NOTES:

1. These are the positions of the control bits in the relevant VMX control fields.

The 0-settings of these VMX controls enable all VMX-specific packet information. The scenarios that would use these default settings also do not require the VMM to use VMX MSR-load areas to enable and disable trace-packet generation across VMX transitions.

If IA32_VMX_MISC[bit 14] reports 0, the 1-settings of the VMX controls in Table 36-52 are not supported, and VM entry will fail on any attempt to set them.

36.5.2 Managing Trace Packet Generation Across VMX Transitions

In tracing scenarios that collect packets for both VMX root operation and VMX non-root operation, a host executive can manage the MSRs associated with trace packet generation directly. The states of these MSRs need not be modified across VMX transitions.

For tracing scenarios that collect packets only within VMX root operation or only within VMX non-root operation, the VMM can toggle IA32_RTIT_CTL.TraceEn on VMX transitions.

36.5.2.1 System-Wide Tracing

When a host or VMM configures Intel PT to collect trace packets of the entire system, it can leave the relevant VMX controls clear to allow VMX-specific packets to provide information across VMX transitions.

The decoder will desire to identify the occurrence of VMX transitions. The packets of interests to a decoder are shown in Table 36-53.

Table 36-53. Packets on VMX Transitions (System-Wide Tracing)

Event	Packets	Enable	Description
VM exit	EVD.VMXR, EVD.VMXQ, CFE.VMEXIT*	EventEn	The CFE identifies the transfer as a VM exit, while the associated EVDs provide the exit reason and exit qualification.
	FUP(GuestIP)	BranchEn or EventEn	The FUP indicates at which point in the guest flow the VM exit occurred. This is important, since VM exit can be an asynchronous event. The IP will match that written into the VMCS.
	PIP(HostCR3, NR=0)		The PIP packet provides the new host CR3 value, as well as indication that the logical processor is entering VMX root operation. This allows the decoder to identify the change of executing context from guest to host and load the appropriate set of binaries to continue decode.
	TIP(HostIP)	BranchEn	The TIP indicates the destination IP, the IP of the first instruction to be executed in VMX root operation. Note, this packet could be preceded by a MODE.Exec packet (Section 36.4.2.8). This is generated only in cases where CS.D or (CS.L & EFER.LMA) change during the transition.
VM entry	CFEVMENTRY, FUP(CLIP)	EventEn	The CFE identifies the transfer as a VM entry, while the FUP identifies the VMLAUNCH/VMRESUME IP.
	PIP(GuestCR3, NR=1)	BranchEn	The PIP packet provides the new guest CR3 value, as well as indication that the logical processor is entering VMX non-root operation. This allows the decoder to identify the change of executing context from host to guest and load the appropriate set of binaries to continue decode.
	TIP(GuestIP)	BranchEn	The TIP indicates the destination IP, the IP of the first instruction to be executed in VMX non-root operation. This should match the RIP loaded from the VMCS. Note, this packet could be preceded by a MODE.Exec packet (Section 36.4.2.8). This is generated only in cases where CS.D or (CS.L & EFER.LMA) change during the transition.

Since the VMX controls that suppress packet generation are cleared, a VMCS packet will be included in all PSB+ for this usage scenario. Additionally, VMPTRLD will generate such a packet. Thus the decoder can distinguish the execution context of different VMs.

When the host VMM configures a system to collect trace packets in this scenario, it should emulate CPUID to report CPUID.07H.00H:EBX[26] as 0 to guests, indicating to guests that Intel PT is not available.

VMX TSC Manipulation

The TSC packets generated while in VMX non-root operation will include any changes resulting from the use of a VMM's use of the TSC offsetting or TSC scaling VMX controls (see Chapter 28, "VMX Non-Root Operation"). In this system-wide usage model, the decoder may need to account for the effect of per-VM adjustments in the TSC packets generated in VMX non-root operation and the absence of TSC adjustments in TSC packets generated in VMX root operation. The VMM can supply this information to the decoder.

36.5.2.2 Guest-Only Tracing

A VMM can configure trace-packet generation while in VMX non-root operation for guests executing normally. This is accomplished by utilizing VMCS controls to manipulate the guest IA32_RTIT_CTL value on VMX transitions. For older processors that do not support these VMCS controls, a VMM can use the VMX MSR-load areas on VM exits (see Section 27.7.2, "VM-Exit Controls for MSRs") and VM entries (see Section 27.8.2, "VM-Entry Controls for MSRs") to limit trace-packet generation to the guest environment.

For this usage, VM entry is programmed to enable trace packet generation, while VM exit is programmed to clear IA32_RTIT_CTL.TraceEn so as to disable trace-packet generation in the host. Further, if it is preferred that the

guest packet stream contain no indication that execution was in VMX non-root operation, the VMM should set to 1 all the VMX controls enumerated in Table 36-52.

36.5.2.3 Emulation of Intel PT Traced State

If a VMM emulates an element of processor state by taking a VM exit on reads and/or writes to that piece of state, and the state element impacts Intel PT packet generation or values, it may be incumbent upon the VMM to insert or modify the output trace data.

If a VM exit is taken on a guest write to CR3 (including “MOV CR3” as well as task switches), the PIP packet normally generated on the CR3 write will be missing.

To avoid decoder confusion when the guest trace is decoded, the VMM should emulate the missing PIP by writing it into the guest output buffer. If the guest CR3 value is manipulated, the VMM may also need to manipulate the IA32_RTIT_CR3_MATCH value, in order to ensure the trace behavior matches the guest's expectation.

Similarly, if a VMM emulates the TSC value by taking a VM exit on RDTSC, the TSC packets generated in the trace may mismatch the TSC values returned by the VMM on RDTSC. To ensure that the trace can be properly aligned with software logs based on RDTSC, the VMM should either make corresponding modifications to the TSC packet values in the guest trace, or use mechanisms such as TSC offsetting or TSC scaling in place of exiting.

36.5.2.4 TSC Scaling

When TSC scaling is enabled for a guest using Intel PT, the VMM should ensure that the value of Maximum Non-Turbo Ratio[15:8] in MSR_PLATFORM_INFO (MSR 0CEH) and the TSC/“core crystal clock” ratio (EBX/EAX) in CPUID.15H are set in a manner consistent with the resulting TSC rate that will be visible to the VM. This will allow the decoder to properly apply TSC packets, MTC packets (based on the core crystal clock or ART, whose frequency is indicated by CPUID.15H), and CBR packets (which indicate the ratio of the processor frequency to the Max Non-Turbo frequency). Absent this, or separate indication of the scaling factor, the decoder will be unable to properly track time in the trace. See Section 36.8.3 for details on tracking time within an Intel PT trace.

36.5.2.5 Failed VM Entry

The packets generated by a failed VM entry depend both on the VMCS configuration, as well as on the type of failure. The results to expect are summarized in the table below. Note that packets in *italics* may or may not be generated, depending on implementation choice, and the point of failure.

Table 36-54. Packets on a Failed VM Entry

Usage Model	Entry Configuration	Early Failure (fall through to next IP)	Late Failure (VM exit like)
System-Wide	No use of “Load IA32_RTIT_CTL” entry control or VM-entry MSR-load area	TIP (NextIP)	<i>CFE.VMENTRY, FUP(CLIP) if EventEn=1</i> <i>PIP(Guest CR3, NR=1), TraceEn 0→1 Packets (See Section 36.2.8.3), PIP(HostCR3, NR=0), TIP(HostIP)</i>
VMM Only	“Load IA32_RTIT_CTL” entry control or VM-entry MSR-load area used to clear TraceEn	TIP (NextIP)	<i>TraceEn 0→1 Packets (See Section 36.2.8.3), TIP(HostIP)</i>
VM Only	“Load IA32_RTIT_CTL” entry control or VM-entry MSR-load area used to set TraceEn	None	None

36.5.2.6 VMX Abort

VMX abort conditions take the processor into a shutdown state. On a VM exit that leads to VMX abort, some packets (FUP, PIP) may be generated, but any expected TIP, TIP.PGE, or TIP.PGD may be dropped.

36.6 TRACING AND SMM TRANSFER MONITOR (STM)

The SMM-transfer monitor (STM) is a VMM that operates inside SMM while in VMX root operation. An STM operates in conjunction with an executive monitor. The latter operates outside SMM and in VMX root operation. Transitions from the executive monitor or its VMs to the STM are called SMM VM exits. The STM returns from SMM via a VM entry to the VM in VMX non-root operation or the executive monitor in VMX root operation.

Intel PT supports tracing in an STM similar to tracing support for VMX operation as described above in Section 36.5. As a result, on a SMM VM exit resulting from #SMI, TraceEn is neither saved nor cleared by default. Software can save the state of the trace configuration MSRs and clear TraceEn using the MSR load/save lists.

Within Event Trace, SMM VM exits generate packets indicating both an #SMI and a VM exit. Similarly, VM entries that return from SMM generate packets that indicate both an RSM and a VM entry. SMM VM exits initiated by the VMCALL instruction do not generate any CFE packet, though the subsequent VM entry returning from SMM will generate a CFE.RSM.

36.7 PACKET GENERATION SCENARIOS

The following tables provides examples of packet generation for various operations. The following acronyms are used in the packet examples below:

- CLIP - Current LIP
- NLIP - Next Sequential LIP
- BLIP - Branch Target LIP

Table 36-55 illustrates the packets generated by a series of example operations, assuming that PacketEn (TriggerEn && ContextEn && FilterEn && BranchEn) is set before and after the operation.

Table 36-55. Packet Generation under Different Example Operations

Case	Operation	Details	Packets
1	Normal non-jump operation		None
2	Conditional branch	6th branch in internal TNT buffer	TNT
3	Conditional branch	1 st ..5 th branch in internal TNT buffer	None
4	Near indirect JMP or CALL		TIP(BLIP)
5	Direct near JMP or CALL		None
6	Near RET	Uncompressed	TIP(BLIP)
7	Near RET	Compressed, 6th branch in internal TNT buffer	TNT
8	Far Branch	Assumes no update to CR3, CS.L, or CS.D	TIP(BLIP)
9	Far Branch	Assumes update to CR3	PIP(NewCR3), TIP(BLIP)
10	Far Branch	Assumes update to CR3 and CS.D/CS.L	PIP(NewCR3), MODE.Exec, TIP(BLIP)
11	External Interrupt or NMI	Assumes no update to CR3, CS.D, or CS.L	FUP(NLIP), TIP(BLIP)
12	External Interrupt or NMI	Assumes update to CR3 and CS.D/CS.L	FUP(NLIP), PIP(NewCR3), MODE.Exec, TIP(BLIP)
13	Exception/Fault or Software Interrupt	Assumes no update to CR3, CS.D, or CS.L	FUP(CLIP), TIP(BLIP)
14	MOV to CR3		PIP(NewCR3, NR)
15	VM exit	Assumes system-wide tracing, see Section 36.5.2.1	See Table 36-53
16	VM entry	Assumes system-wide tracing, see Section 36.5.2.1	See Table 36-53

Table 36-55. Packet Generation under Different Example Operations

Case	Operation	Details	Packets
17	ENCLU[EENTER] / ENCLU[ERESUME] / ENCLU[EEXIT] / AEX/EEE	Only debug enclaves allow PacketEn to be set during enclave execution. Assumes no change to CS.L or CS.D.	FUP(CLIP), TIP(BLIP)
18	XBEGIN/XACQUIRE/XEND/XRELEASE	Does not begin/end transactional execution	None
19	XBEGIN/XACQUIRE	Assumes beginning of transactional execution	MODE.TSX(InTX=1, TXAbort=0), FUP(CLIP)
20	XEND/XRELEASE	Completes transaction	MODE.TSX(InTX=0, TXAbort=0), FUP(CLIP)
21	XABORT or Asynchronous Abort	Aborts transactional execution	MODE.TSX(InTX=0, TXAbort=1), FUP(CLIP), TIP(BLIP)
22	INIT	On BSP. Assumes no CR3, CS.D, or CS.L update.	FUP(NLIP), TIP(ResetLIP)
23	INIT	On AP, goes to wait-for-SIPI. Assumes no CR3 update.	FUP(NLIP)
24	SIPI	Assumes no CS.D or CS.L update	TIP.PGE(SIPI.LIP)
25	Wake from state deeper than C0.1, P-state change, or other scenario where timing packets (MTC, CYC) may have ceased.	TSC if TSCEn=1 TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR
26	UINTR	User interrupt handler entry.	FUP(NLIP), TIP(BLIP)
27	UIRET	Exiting from user interrupt handler.	TIP(BLIP)

Table 36-56 illustrates the packets generated in example scenarios where the operation alters the value of PacketEn. Note that insertion of PSB+ is not included here, though it can be coincident with initial enabling of Intel PT. See Section 36.3.7 for details.

Table 36-56. Packet Generation with Operations That Alter the Value of PacketEn

Case	Operation	PktEn Before	PktEn After	CntxEn After	Details	Packets
1	WRMSR/XRSTORS that changes TraceEn 0 → 1	0	1	0	TSC if TSCEn=1; TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR, MODE.Exec
2	WRMSR/XRSTORS that changes TraceEn 0 → 1	0	1	1	TSC if TSCEn=1; TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR, MODE.Exec, TIP.PGE(NLIP)
3	WRMSR that changes TraceEn 1 → 0	1	0	D.C.		FUP(CLIP), TIP.PGD()
4	Taken Branch	1	0	1	Source is in IP filter region. Target is outside IP filter region.	TIP.PGD(BLIP)
5	Taken Branch, Interrupt, EEXIT, etc.	0	1	1	Source is outside IP filter region. Target is in IP filter region.	TIP.PGE(BLIP)
6	Far Branch, Interrupt, EENTER, etc.	1	0	0	Requires change to CPL or CR3, or entry to opt-out enclave.	TIP.PGD()
7	Trap-like event (external interrupt, NMI, VM exit/entry, etc.)	1	0	0	Requires change to CPL or CR3.	FUP(NLIP), TIP.PGD()
8	Fault-like event (exception/fault, software interrupt, VM exit/entry, etc.)	1	0	0	Requires change to CPL or CR3.	FUP(CLIP), TIP.PGD()

Table 36-56. Packet Generation with Operations That Alter the Value of PacketEn (Contd.)

Case	Operation	PktEn Before	PktEn After	CntxEn After	Details	Packets
9	SMI, VM exit/entry	1	0	0	TraceEn is cleared.	FUP(NLIP), TIP.PGD()
10	RSM, VM exit/entry	0	1	1	TraceEn is set.	See Case 2 for packets on enable. FUP/TIP.PGE IP is the BLIP.
11	VM Exit	1	0	0	Assumes guest-only tracing, see Section 36.5.2.2. TraceEn is cleared.	FUP(VMCSg.RIP), TIP.PGD()
12	VM entry	0	1	1	Assumes guest-only tracing, see Section 36.5.2.2. TraceEn is set.	TIP.PGE(VMCSg.RIP)

Table 36-57 illustrates examples of PTWRITE, assuming TriggerEn && PTWEn is true.

Table 36-57. Examples of PTWRITE when TriggerEn && PTWEn is True

Case	Operation	ContextEn	Details	Packets
1	MWAIT/UMWAIT gets fault or VM exit.	D.C.		None. Other trace sources may generate packets on fault or VM exit.
2	MWAIT/UMWAIT requests C0, or monitor not armed, or VMX virtual-interrupt delivery.	D.C.		None.
3	MWAIT/UMWAIT enters C-state deeper than C0.1.	0		PWRE(Cx), EXSTOP
4	MWAIT/UMWAIT enters C-state deeper than C0.1.	1		MWAIT(Cy), PWRE(Cx), EXSTOP(IP), FUP(CLIP)
5	HLT, Triple-fault shutdown, other operation that enters C1.	1		PWRE(C1), EXSTOP(IP), FUP(CLIP)
6	Hardware Duty Cycling (HDC).	1	TSC if TSCEn=1 TMA if TSCEn=MTCEn=1	PWRE(HW, C6), EXSTOP(IP), FUP(NLIP), TSC?, TMA?, CBR, PWRX(CC6, CC6, 0x8)
7	Wake event during Cx (x > 0).	D.C.	TSC if TSCEn=1 TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR, PWRX(LCC, DCC, 0x1) Other trace sources may generate packets for the wake operation (e.g., interrupt).

Table 36-58 illustrates examples of Power Event Trace, assuming TriggerEn && PwrEvtEn is true.

Table 36-58. Examples of Power Event Trace when TriggerEn && PwrEvtEn is True

Case	Operation	ContextEn && FilterEn	Details	Packets
1	PTWRITE rm32/64	0		None
2	PTWRITE rm32	1	FUP, PTW.IP=1 if FUPonPTW=1	PTW(IP=1?, 4B, rm32_value), FUP(CLIP)?
3	PTWRITE rm64	1	FUP, PTW.IP=1 if FUPonPTW=1	PTW(IP=1?, 8B, rm64_value), FUP(CLIP)?

Table 36-59 illustrates examples of Event Trace, assuming TriggerEn && ContextEn && EventEn is true. In all cases, other trace sources (e.g., BranchEn), if enabled, may generate additional packets. For details, see the other tables in this section.

Table 36-59. Event Trace Examples when TriggerEn && ContextEn && EventEn is True

Case	Operation	ContextEn Before	ContextEn After	Details	Packets
1	IRET, ERETS, ERETU	1	D.C.		CFE.IRET(IP=1), FUP(CLIP)
2	IRET, ERETS, ERETU	0	1		CFE.IRET
3	External interrupt, including NMI	1	D.C.		CFE.INTR(IP=1, Vector), FUP(NLIP)
4	External interrupt, including NMI	1	1	Assumes BranchEn=1, illustrates the shared FUP.	CFE.INTR(IP=0, Vector), FUP(NLIP), TIP(BLIP)
5	SW Interrupt (without FRED), Exception/Fault other than #PF	1	D.C.		CFE.INTR(IP=1, Vector), FUP(CLIP)
6	Page Fault (#PF)	1	D.C.		EVD.PFA, CFE.INTR(IP=1,14), FUP(CLIP)
7	Page Fault (#PF)	0	D.C.		None
10	SMI	1	D.C.		CFE.SMI(IP=1), FUP(NLIP)
11	RSM, TraceEn restored to 1	D.C.	1		CFE.RSM(IP=0)
12	Entry to Shutdown	1	D.C.		CFE.SHUTDOWN(IP=1), FUP(CLIP)
13	VM exit caused by interrupt, fault, or SMI	1	D.C.	Assumes “Conceal VMX in PT” exit control is 0.	EVD.VMXQ, EVD.VMXR, CFE.VMEXIT_IT_INTR(IP=1, Vector), FUP(VMCSg.LIP)
14	VM exit caused by other than interrupt, fault, or SMI	1	D.C.	Assumes “Conceal VMX in PT” exit control is 0.	EVD.VMXQ, EVD.VMXR, CFE.VMEXIT(IP=1), FUP(VMCSg.LIP)
15	VM exit caused by other than interrupt, fault, or SMI	0	1	Assumes “Conceal VMX in PT” exit control is 0.	CFE.VMEXIT(IP=0)
16	VM entry	1	D.C.	Assumes “Conceal VMX in PT” entry control is 0.	CFE.VMENTRY(IP=1), FUP(VMCSg.LIP)
17	AEX/EEE, from opt-out (non-debug) enclave	0	0		None
18	AEX/EEE, from opt-out (non-debug) enclave	0	1		CFE.INTR(IP=0)
19	AEX, from opt-in (debug) enclave	1	D.C.		CFE.INTR(IP=1, Vec), FUP(AEP LIP)
20	INIT	1	D.C.		CFE.INIT(IP=1), FUP(NLIP)
21	SIPI	1	D.C.		CFE.SIPI(IP=0)
22	STI/CLI/POPF	1	1	Assumes a change to RFLAGS.IF.	MODE.Exec, FUP(NLIP)
23	SW Interrupt (with FRED)	1	D.C.		CFE.SWINTR(IP=1, Vector), FUP(CLIP)
24	SYSCALL, SYSENTER (with FRED)	1	D.C.		CFE.SYSCALL(IP=1), FUP(CLIP)

36.8 SOFTWARE CONSIDERATIONS

36.8.1 Tracing SMM Code

Nothing prevents an SMM handler from configuring and enabling packet generation for its own use. As described in Section 36.2.9.3, SMI will always clear TraceEn, so the SMM handler would have to set TraceEn in order to enable tracing. There are some unique aspects and guidelines involved with tracing SMM code, which follow:

1. SMM should save away the existing values of any configuration MSRs that SMM intends to modify for tracing. This will allow the non-SMM tracing context to be restored before RSM.
2. It is recommended that SMM wait until it sets CSbase to 0 before enabling packet generation, to avoid possible LIP vs RIP confusion.
3. Packet output cannot be directed to SMRR memory, even while tracing in SMM.
4. Before performing RSM, SMM should take care to restore modified configuration MSRs to the values they had immediately after #SMI. This involves first disabling packet generation by clearing TraceEn, then restoring any other configuration MSRs that were modified.
5. RSM
 - Software must ensure that TraceEn=0 at the time of RSM. Tracing RSM is not a supported usage model, and the packets generated by RSM are undefined.
 - For processors on which Intel PT and LBR use are mutually exclusive (see Section 36.3.1.2), any RSM during which TraceEn is restored to 1 will suspend any LBR or BTS logging.

36.8.2 Cooperative Transition of Multiple Trace Collection Agents

A third-party trace-collection tool should take into consideration the fact that it may be deployed on a processor that supports Intel PT but may run under any operating system.

In such a deployment scenario, Intel recommends that tool agents follow similar principles of cooperative transition of single-use hardware resources, similar to how performance monitoring tools handle performance monitoring hardware:

- Respect the “in-use” ownership of an agent who already configured the trace configuration MSRs, see architectural MSRs with the prefix “IA32_RTIT_” in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, where “in-use” can be determined by reading the “enable bits” in the configuration MSRs.
- Relinquish ownership of the trace configuration MSRs by clearing the “enabled bits” of those configuration MSRs.

36.8.3 Tracking Time

This section describes the relationships of several clock counters whose update frequencies reside in different domains that feed into the timing packets. To track time, the decoder also needs to know the regularity or irregularity of the occurrences of various timing packets that store those clock counters.

Intel PT provides time information for three different but related domains:

- Processor timestamp counter

This counter increments at the max non-turbo or P1 frequency, and its value is returned on a RDTSC. Its frequency is fixed. The TSC packet holds the lower 7 bytes of the timestamp counter value. The TSC packet occurs occasionally and are much less frequent than the frequency of the time stamp counter. The timestamp counter will continue to increment when the processor is in deep C-States, with the exception of processors reporting CPUID.80000007H:EDX.TSC_INVARIANT[8] =0.

- Core crystal clock

The ratio of the core crystal clock to timestamp counter frequency is known as P, and can be calculated as CPUID.15H:EBX[31:0] / CPUID.15H:EAX[31:0]. The frequency of the core crystal clock is fixed and lower than that of the timestamp counter. The periodic MTC packet is generated based on software-selected

multiples of the crystal clock frequency. The MTC packet is expected to occur more frequently than the TSC packet.

- Processor core clock

The processor core clock frequency can vary due to P-state and thermal conditions. The CYC packet provides elapsed time as measured in processor core clock cycles relative to the last CYC packet.

A decoder can use all or some combination of these packets to track time at different resolutions throughout the trace packets.

36.8.3.1 Time Domain Relationships

The three domains are related by the following formula:

$$\text{TimeStampValue} = (\text{CoreCrystalClockValue} * P) + \text{AdjustedProcessorCycles} + \text{Software\_Offset};$$

The CoreCrystalClockValue, also known as the Always Running Timer (ART) value, can provide the coarse-grained component of the TSC value. P, or the TSC/ART ratio, can be derived from CPUID.15H, as described in Section 36.8.3.

The AdjustedProcessorCycles component provides the fine-grained distance from the rising edge of the last core crystal clock. Specifically, it is a cycle count in the same frequency as the timestamp counter from the last crystal clock rising edge. The value is adjusted based on the ratio of the processor core clock frequency to the Maximum Non-Turbo (or P1) frequency.

The Software_Offsets component includes software offsets that are factored into the timestamp value, such as IA32_TSC_ADJUST.

36.8.3.2 Estimating TSC within Intel PT

For many usages, it may be useful to have an estimated timestamp value for all points in the trace. The formula provided in Section 36.8.3.1 above provides the framework for how such an estimate can be calculated from the various timing packets present in the trace.

The TSC packet provides the precise timestamp value at the time it is generated; however, TSC packets are infrequent, and estimates of the current timestamp value based purely on TSC packets are likely to be very inaccurate for this reason. In order to get more precise timing information between TSC packets, CYC packets and/or MTC packets should be enabled.

MTC packets provide incremental updates of the CoreCrystalClockValue. On processors that support CPUID.15H, the frequency of the timestamp counter and the core crystal clock is fixed, thus MTC packets provide a means to update the running timestamp estimate. Between two MTC packets A and B, the number of crystal clock cycles passed is calculated from the 8-bit payloads of respective MTC packets:

$$(\text{CTC}_B - \text{CTC}_A), \text{ where } \text{CTC}_i = \text{MTC}_i[15:8] \ll \text{IA32\_RTIT\_CTL.MTCFreq} \text{ and } i = A, B.$$

The time from a TSC packet to the subsequent MTC packet can be calculated using the TMA packet that follows the TSC packet. The TMA packet provides both the crystal clock value (lower 16 bits, in the CTC field) and the AdjustedProcessorCycles value (in the FastCounter field) that can be used in the calculation of the corresponding core crystal clock value of the TSC packet.

When the next MTC after a pair of TSC/TMA is seen, the number of crystal clocks passed since the TSC packet can be calculated by subtracting the TMA.CTC value from the time indicated by the MTC_{Next} packet by

$$\text{CTC}_{\text{Delta}}[15:0] = (\text{CTC}_{\text{Next}}[15:0] - \text{TMA.CTC}[15:0]), \text{ where } \text{CTC}_{\text{Next}} = \text{MTC}_{\text{Payload}} \ll \text{IA32\_RTIT\_CTL.MTCFreq}.$$

The TMA.FastCounter field provides the number of AdjustedProcessorCycles since the last crystal clock rising edge, from which it can be determined the percentage of the next crystal clock cycle that had passed at the time of the TSC packet.

CYC packets can provide further precision of an estimated timestamp value to many non-timing packets, by providing an indication of the time passed between other timing packets (MTCs or TSCs).

When enabled, CYC packets are sent preceding each CYC-eligible packet, and provide the number of processor core clock cycles that have passed since the last CYC packet. Thus between MTCs and TSCs, the accumulated CYC values can be used to estimate the AdjustedProcessorCycles component of the timestamp value. The accumulated CPU cycles will have to be adjusted to account for the difference in frequency between the processor core clock and

the P1 frequency. The necessary adjustment can be estimated using the core:bus ratio value given in the CBR packet, by multiplying the accumulated cycle count value by $P1/CBR_{\text{payload}}$.

Note that stand-alone TSC packets (that is, TSC packets that are not a part of a PSB+) are typically generated only when generation of other timing packets (MTCs and CYCs) has ceased for a period of time. Example scenarios include when Intel PT is re-enabled, or on wake after a sleep state. Thus any calculated estimate of the timestamp value leading up to a TSC packet will likely result in a discrepancy, which the TSC packet serves to correct.

A greater level of precision may be achieved by calculating the CPU clock frequency, see Section 36.8.3.4 below for a method to do so using Intel PT packets.

CYCs can be used to estimate time between TSCs even without MTCs, though this will likely result in a reduction in estimated TSC precision.

36.8.3.3 VMX TSC Manipulation

When software executes in non-Root operation, additional offset and scaling factors may be applied to the TSC value. These are optional, but may be enabled via VMCS controls on a per-VM basis. See Chapter 28, “VMX Non-Root Operation,” for details on VMX TSC offsetting and TSC scaling.

Like the value returned by RDTSC, TSC packets will include these adjustments, but other timing packets (such as MTC, CYC, and CBR) are not impacted. In order to use the algorithm above to estimate the TSC value when TSC scaling is in use, it will be necessary for software to account for the scaling factor. See Section 36.5.2.4 for details.

36.8.3.4 Calculating Frequency with Intel PT

Because Intel PT can provide both wall-clock time and processor clock cycle time, it can be used to measure the processor core clock frequency. Either TSC or MTC packets can be used to track the wall-clock time. By using CYC packets to count the number of processor core cycles that pass in between a pair of wall-clock time packets, the ratio between processor core clock frequency and TSC frequency can be derived. If the P1 frequency is known, it can be applied to determine the CPU frequency. See Section 36.8.3.1 above for details on the relationship between TSC, MTC, and CYC.

29. Updates to Chapter 37, Volume 3D

Change bars and text show changes to Chapter 37 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- This chapter was subsequently re-numbered after inserting a new chapter 8 for Flexible Return and Event Delivery (FRED).
- Replaced Table 37-4, Table 37-5, and Table 37-6 with references to Chapter 21 of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

CHAPTER 37

INTRODUCTION TO INTEL® SOFTWARE GUARD EXTENSIONS

37.1 OVERVIEW

Intel® Software Guard Extensions (Intel® SGX) is a set of instructions and mechanisms for memory accesses added to Intel® Architecture processors. Intel SGX can encompass two collections of instruction extensions, referred to as SGX1 and SGX2, see Table 37-1 and Table 37-2. The SGX1 extensions allow an application to instantiate a protected container, referred to as an enclave. The enclave is a trusted area of memory, where critical aspects of the application functionality have hardware-enhanced confidentiality and integrity protections. New access controls to restrict access to software not resident in the enclave are also introduced. The SGX2 extensions allow additional flexibility in runtime management of enclave resources and thread execution within an enclave.

Chapter 38 covers main concepts, objects and data structure formats that interact within the Intel SGX architecture. Chapter 39 covers operational aspects ranging from preparing an enclave, transferring control to enclave code, and programming considerations for the enclave code and system software providing support for enclave execution. Chapter 40 describes the behavior of Asynchronous Enclave Exit (AEX) caused by events while executing enclave code. Chapter 41 covers the syntax and operational details of the instruction and associated leaf functions available in Intel SGX. Chapter 42 describes interaction of various aspects of IA32 and Intel® 64 architectures with Intel SGX. Chapter 43 covers Intel SGX support for application debug, profiling, and performance monitoring.

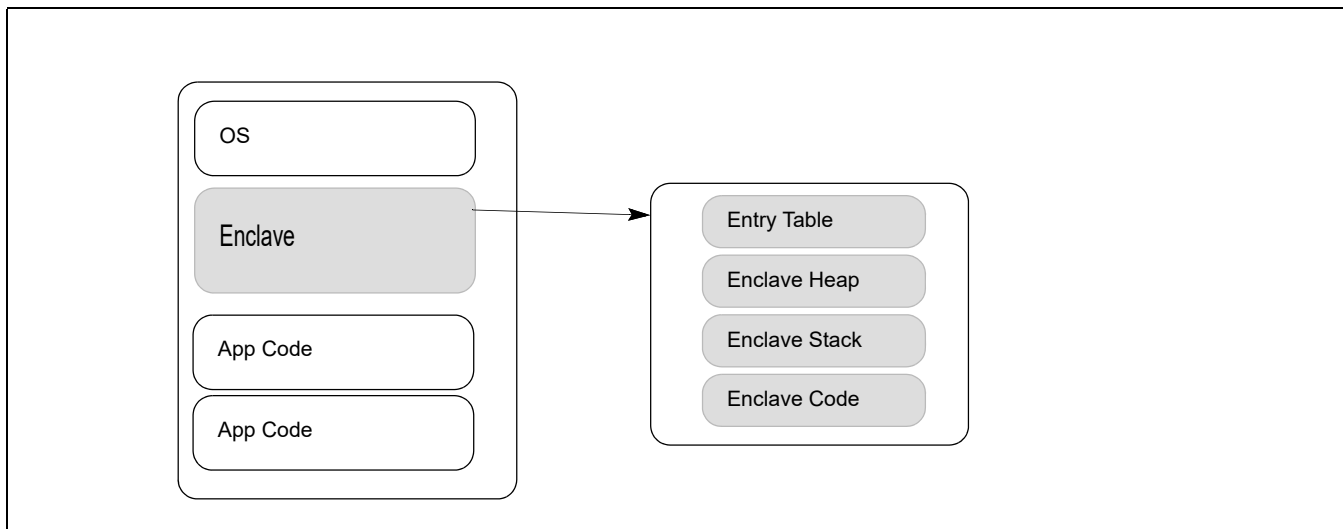


Figure 37-1. An Enclave Within the Application's Virtual Address Space

37.2 ENCLAVE INTERACTION AND PROTECTION

Intel SGX allows the protected portion of an application to be distributed in the clear. Before the enclave is built, the enclave code and data are free for inspection and analysis. The protected portion is loaded into an enclave where its code and data is measured. Once the application's protected portion of the code and data are loaded into an enclave, memory access controls are in place to restrict access by external software. An enclave can prove its identity to a remote party and provide the necessary building-blocks for secure provisioning of keys and credentials. The application can also request an enclave-specific and platform-specific key that it can use to protect keys and data that it wishes to store outside the enclave.¹

1. For additional information, see white papers on Intel SGX at <https://www.intel.com/content/www/us/en/developer/tools/isa-extensions/overview.html>.

Intel SGX introduces two significant capabilities to the Intel Architecture. First is the change in enclave memory access semantics. The second is protection of the address mappings of the application.

37.3 ENCLAVE LIFE CYCLE

Enclave memory management is divided into two parts: address space allocation and memory commitment. Address space allocation is the specification of the range of linear addresses that the enclave may use. This range is called the ELRANGE. No actual resources are committed to this region. Memory commitment is the assignment of actual memory resources (as pages) within the allocated address space. This two-phase technique allows flexibility for enclaves to control their memory usage and to adjust dynamically without overusing memory resources when enclave needs are low. Commitment adds physical pages to the enclave. An operating system may support separate allocate and commit operations.

During enclave creation, code and data for an enclave are loaded from a clear-text source, i.e., from non-enclave memory.

Untrusted application code starts using an initialized enclave typically by using the EENTER leaf function provided by Intel SGX to transfer control to the enclave code residing in the protected Enclave Page Cache (EPC). The enclave code returns to the caller via the EEXIT leaf function. Upon enclave entry, control is transferred by hardware to software inside the enclave. The software inside the enclave switches the stack pointer to one inside the enclave. When returning back from the enclave, the software swaps back the stack pointer then executes the EEXIT leaf function.

On processors that support the SGX2 extensions, an enclave writer may add memory to an enclave using the SGX2 instruction set, after the enclave is built and running. These instructions allow adding additional memory resources to the enclave for use in such areas as the heap. In addition, SGX2 instructions allow the enclave to add new threads to the enclave. The SGX2 features provide additional capabilities to the software model without changing the security properties of the Intel SGX architecture.

Calling an external procedure from an enclave could be done using the EEXIT leaf function. Software would use EEXIT and a software convention between the trusted section and the untrusted section.

An active enclave consumes resources from the Enclave Page Cache (EPC, see Section 37.5). Intel SGX provides the EREMOVE instruction that an EPC manager can use to reclaim EPC pages committed to an enclave. The EPC manager uses EREMOVE on every enclave page when the enclave is torn down. After successful execution of EREMOVE the EPC page is available for allocation to another enclave.

37.4 DATA STRUCTURES AND ENCLAVE OPERATION

There are 2 main data structures associated with operating an enclave, the SGX Enclave Control Structure (SECS, see Section 38.7) and the Thread Control Structure (TCS, see Section 38.8).

There is one SECS for each enclave. The SECS contains meta-data about the enclave which is used by the hardware and cannot be directly accessed by software. Included in the SECS is a field that stores the enclave build measurement value. This field, MRENCLAVE, is initialized by the ECREATE instruction and updated by every EADD and EEXTEND. It is locked by EINIT.

Every enclave contains one or more TCS structures. The TCS contains meta-data used by the hardware to save and restore thread specific information when entering/exiting the enclave. There is one field, FLAGS, that may be accessed by software. This field can only be accessed by debug enclaves. The flag bit, DBGOPTIN, allows to single step into the thread associated with the TCS. (see Section 38.8.1)

The SECS is created when ECREATE (see Table 37-1) is executed. The TCS can be created using the EADD instruction or the SGX2 instructions (see Table 37-2).

37.5 ENCLAVE PAGE CACHE

The Enclave Page Cache (EPC) is the secure storage used to store enclave pages when they are a part of an executing enclave. For an EPC page, hardware performs additional access control checks to restrict access to the page. After the current page access checks and translations are performed, the hardware checks that the EPC page

is accessible to the program currently executing. Generally an EPC page is only accessed by the owner of the executing enclave or an instruction which is setting up an EPC page

The EPC is divided into EPC pages. An EPC page is 4KB in size and always aligned on a 4KB boundary.

Pages in the EPC can either be valid or invalid. Every valid page in the EPC belongs to one enclave instance. Each enclave instance has an EPC page that holds its SECS. The security metadata for each EPC page is held in an internal micro-architectural structure called Enclave Page Cache Map (EPCM, see Section 37.5.1).

The EPC is managed by privileged software. Intel SGX provides a set of instructions for adding and removing content to and from the EPC. The EPC may be configured by BIOS at boot time. On implementations in which EPC memory is part of system DRAM, the contents of the EPC are protected by an encryption engine.

37.5.1 Enclave Page Cache Map (EPCM)

The EPCM is a secure structure used by the processor to track the contents of the EPC. The EPCM holds one entry for each page in the EPC. The format of the EPCM is micro-architectural, and consequently is implementation dependent. However, the EPCM contains the following architectural information:

- The status of EPC page with respect to validity and accessibility.
- An SECS identifier (see Section 38.20) of the enclave to which the page belongs.
- The type of page: regular, SECS, TCS or VA.
- The linear address through which the enclave is allowed to access the page.
- The specified read/write/execute permissions on that page.

The EPCM structure is used by the CPU in the address-translation flow to enforce access-control on the EPC pages. The EPCM structure is described in Table 38-28, and the conceptual access-control flow is described in Section 38.5.

The EPCM entries are managed by the processor as part of various instruction flows.

37.6 ENCLAVE INSTRUCTIONS AND INTEL® SGX

The enclave instructions available with Intel SGX are organized as leaf functions under three instruction mnemonics: ENCLS (ring 0) and ENCLU (ring 3). Each leaf function uses EAX to specify the leaf function index, and may require additional implicit input registers as parameters. The use of EAX is implied implicitly by the ENCLS and ENCLU instructions; ModR/M byte encoding is not used with ENCLS and ENCLU. The use of additional registers does not use ModR/M encoding and is implied implicitly by the respective leaf function index.

Each leaf function index is also associated with a unique, leaf-specific mnemonic. A long-form expression of Intel SGX instruction takes the form of ENCLx[LEAF_MNEMONIC], where 'x' is either 'S', 'U', or 'V'. The long-form expression provides clear association of the privilege-level requirement of a given "leaf mnemonic". For simplicity, the unique "Leaf_Mnemonic" name is used (omitting the ENCLx for convenience) throughout in this document.

Details of individual SGX leaf functions are described in Chapter 41. Table 37-1 provides a summary of the instruction leaves that are available in the initial implementation of Intel SGX, which is introduced in the 6th generation Intel Core processors. Table 37-2 summarizes enhancement of Intel SGX for future Intel processors.

Table 37-1. Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX1

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EADD]	Add an EPC page to an enclave.	ENCLU[EENTER]	Enter an enclave.
ENCLS[EBLOCK]	Block an EPC page.	ENCLU[EEXIT]	Exit an enclave.
ENCLS[ECREATE]	Create an enclave.	ENCLU[EGETKEY]	Create a cryptographic key.
ENCLS[EDBGDR]	Read data from a debug enclave by debugger.	ENCLU[EREPORT]	Create a cryptographic report.
ENCLS[EDBGWR]	Write data into a debug enclave by debugger.	ENCLU[ERESUME]	Re-enter an enclave.

Table 37-1. Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX1

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EEXTEND]	Extend EPC page measurement.		
ENCLS[EINIT]	Initialize an enclave.		
ENCLS[ELDB]	Load an EPC page in blocked state.		
ENCLS[ELDU]	Load an EPC page in unblocked state.		
ENCLS[EPA]	Add an EPC page to create a version array.		
ENCLS[EREMOVE]	Remove an EPC page from an enclave.		
ENCLS[ETRACK]	Activate EBLOCK checks.		
ENCLS[EWB]	Write back/invalidate an EPC page.		

Table 37-2. Supervisor and User Mode Enclave Instruction Leaf Functions in Long-Form of SGX2

Supervisor Instruction	Description	User Instruction	Description
ENCLS[EAUG]	Allocate EPC page to an existing enclave.	ENCLU[EACCEPT]	Accept EPC page into the enclave.
ENCLS[EMODPR]	Restrict page permissions.	ENCLU[EMODPE]	Enhance page permissions.
ENCLS[EMODT]	Modify EPC page type.	ENCLU[EACCEPTCOPY]	Copy contents to an augmented EPC page and accept the EPC page into the enclave.

37.7 DISCOVERING SUPPORT FOR INTEL® SGX AND ENABLING ENCLAVE INSTRUCTIONS

Detection of support of Intel SGX and enumeration of available and enabled Intel SGX resources are queried using the CPUID instruction. The enumeration interface comprises the following:

- Processor support of Intel SGX is enumerated by a feature flag in CPUID.07H: CPUID.07H.00H:EBX.SGX[2]. If CPUID.07H.00H:EBX.SGX = 1, the processor has support for Intel SGX, and requires opt-in enabling by BIOS via IA32_FEATURE_CONTROL MSR.

If CPUID.07H.00H:EBX.SGX = 1, CPUID will report via the available sub-leaves of CPUID.12H on available and/or configured Intel SGX resources.

- The available and configured Intel SGX resources enumerated by the sub-leaves of CPUID.12H depend on the state of BIOS configuration.

37.7.1 Intel® SGX Opt-In Configuration

On processors that support Intel SGX, IA32_FEATURE_CONTROL provides the SGX_ENABLE field (bit 18). Before system software can configure and enable Intel SGX resources, BIOS is required to set IA32_FEATURE_CONTROL.SGX_ENABLE = 1 to opt-in the use of Intel SGX by system software.

The semantics of setting SGX_ENABLE follows the rules of IA32_FEATURE_CONTROL.LOCK (bit 0). Software is considered to have opted into Intel SGX if and only if IA32_FEATURE_CONTROL.SGX_ENABLE and IA32_FEATURE_CONTROL.LOCK are set to 1. The setting of IA32_FEATURE_CONTROL.SGX_ENABLE (bit 18) is not reflected by CPUID.

Table 37-3. Intel® SGX Opt-in and Enabling Behavior

CPUID.07H.00H:EBX.SGX	CPUID.12H	FEATURE_CONTROL.LOCK	FEATURE_CONTROL.SGX_ENABLE	Enclave Instruction
0	Invalid	X	X	#UD
1	Valid*	X	X	#UD**

Table 37-3. Intel® SGX Opt-in and Enabling Behavior

CPUID.07H.00H:EBX.SGX	CPUID.12H	FEATURE_CONTROL.LOCK	FEATURE_CONTROL.SGX_ENABLE	Enclave Instruction
1	Valid*	0	X	#GP
1	Valid*	1	0	#GP
1	Valid*	1	1	Available***
* Leaf 12H enumeration results are dependent on enablement.				
** See list of conditions in the #UD section of the reference pages of ENCLS and ENCLU.				
*** See CPUID.12H in chapter 21 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1 for details of SGX1 and SGX2.				

37.7.2 Intel® SGX Resource Enumeration Leaves

If CPUID.07H.00H:EBX.SGX = 1, the processor also supports querying CPUID.12H on Intel SGX resource capability and configuration. The number of available sub-leaves in leaf 12H depends on the Opt-in and system software configuration. Information returned by CPUID.12H is thread specific; software should not assume that if Intel SGX instructions are supported on one hardware thread, they are also supported elsewhere.

A properly configured processor exposes Intel SGX functionality with CPUID.12H reporting valid information (non-zero content) in three or more sub-leaves, see [CPUID.12H in Chapter 21 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1](#).

- CPUID.12H.00H enumerates Intel SGX capability, including enclave instruction opcode support (see [CPUID.12H in Chapter 21 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1](#)).
- CPUID.12H.01H enumerates Intel SGX capability of processor state configuration and enclave configuration in the SECS structure (see [CPUID.12H in Chapter 21 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1](#)).
- CPUID.(12H, ECX > 1) enumerates available EPC resources.

On processors that support Intel SGX1 and SGX2, CPUID.12H sub-leaf 2 report physical memory resources available for use with Intel SGX. These physical memory sections are typically allocated by BIOS as **Processor Reserved Memory**, and available to the OS to manage as EPC.

To enumerate how many EPC sections are available to the EPC manager, software can enumerate CPUID.12H with sub-leaf index starting from 2, and decode the sub-leaf-type encoding (returned in EAX[3:0]) until the sub-leaf type is invalid. All invalid sub-leaves of CPUID.12H return EAX/EBX/ECX/EDX with 0. See [CPUID.12H in Chapter 21 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1](#).

37.8 INTEL® SGX INTERACTIONS WITH CONTROL-FLOW ENFORCEMENT TECHNOLOGY

This section discusses extensions to the Intel SGX architecture to support CET.

37.8.1 CET in Enclaves Model

Each enclave has its private configuration for CET that is not shared with the CET configurations of the enclosing application. On entry into the enclave, the CET state of the enclosing application is saved into scratchpad registers inside the processor and the CET state of the enclave is established. On an asynchronous exit, the enclave CET state is saved into the enclave state save area frame. On exit from the enclave, the CET state of the enclosing application is re-established from the scratchpad registers.

A new page type, PT_SS_FIRST, is used to denote pages in an enclave that can be used as a first page of a shadow stack.

A new page type, PT_SS_REST, is used to denote pages in an enclave that can be used as a non-first page of a shadow stack.

A page denoted as PT_SS_FIRST and PT_SS_REST will be a legal target for shadow_stack_load, shadow_stack_store, and regular load operations. Regular stores will be disallowed to such pages. A PT_SS_FIRST/PT_SS_REST page must be writeable in the IA page tables and in EPT.

When in enclave mode, shadow_stack_load and shadow_stack_store operations must be to addresses in the enclave ELRANGE.

The EAUG instruction is extended to allocate pages of type PT_SS_FIRST/PT_SS_REST; this page type requires specifying a SECINFO structure with page parameters. Shadow page permission must be R/W. Regular R/W pages may continue to be allocated by providing a SECINFO pointer value of 0. Regular R/W pages may also be allocated by providing a SECINFO structure that specifies the page parameters. The EAUG instruction creates a shadow-stack-restore token at offset 0xFF8 on a PT_SS_FIRST page. This allows a dynamically created shadow stack to be restored using the RSTORSSP instruction. The EADD and EAUG instructions disallow creation of a PT_SS_FIRST or PT_SS_REST page as the first or last page in ELRANGE.

The EADD instruction requires that the PT_SS_REST page be all zeroes. The EADD instruction requires that a PT_SS_FIRST page be all zeroes except the 8 bytes at offset 0xFF8 on that page that must have a shadow-stack-restore token. This shadow-stack-restore token must have a linear address which is the linear address of the PT_SS_FIRST page + 4096. As an enclave could be loaded at varying linear addresses, the enclave builder should not extend the measurement of the PT_SS_FIRST pages into the measurement registers. On first entry on to the enclave using a TCS, the enclave software can use the RSTORSSP instruction to restore its SSP. Subsequent to performing a RSTORSSP, the enclave software can use the INCSSP instruction to pop the previous-ssp token that is created by the RSTORSSP instruction at the top of the restored shadow stack.

On an enclave entry, the SSP will be initialized to the value in a new TCS field called PREVSSP. The PREVSSP field is written with the value of SSP on enclave exit and is loaded into SSP at enclave entry. When a TCS page is added using EADD or accepted using EACCEPT, the processor requires the PREVSSP field to be initialized to 0.

37.8.2 Operations Not Supported on Shadow Stack Pages

The following operations are not allowed on pages of type PT_SS_FIRST and PT_SS_REST:

- EACCEPTCOPY
- EMODPR
- EMODPE

37.8.3 Indirect Branch Tracking - Legacy Compatibility Treatment

The legacy code page bitmap is tested using the page offset within the ELRANGE instead of the absolute linear address of the address where ENDBRANCH was missed; see the detailed algorithm in Section 18.3.6, "Legacy Compatibility Treatment," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for additional details.

30. Updates Appendix A, Volume 3D

Change bars and text show changes to Appendix A of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- Corrected MAXPHYADDR in footnote 2 on page A-1.
- Added bit 58 for VMX nested-exception support in Section A.1, "Basic VMX Information".
- Updated list of reserved bits in Section A.1.

APPENDIX A

VMX CAPABILITY REPORTING FACILITY

The ability of a processor to support VMX operation and related instructions is indicated by `CPUID.01H:ECX.VMX[5] = 1`. A value 1 in this bit indicates support for VMX features.

Support for specific features detailed in Chapter 29 and other VMX chapters is determined by reading values from a set of capability MSR. These MSRs are indexed starting at MSR address 480H. VMX capability MSRs are read-only; an attempt to write them (with `WRMSR`) produces a general-protection exception (`#GP(0)`). They do not exist on processors that do not support VMX operation; an attempt to read them (with `RDMR`) on such processors produces a general-protection exception (`#GP(0)`).

A.1 BASIC VMX INFORMATION

The `IA32_VMX_BASIC` MSR (index 480H) consists of the following fields:

- Bits 30:0 contain the 31-bit VMCS revision identifier used by the processor. Processors that use the same VMCS revision identifier use the same size for VMCS regions (see subsequent item on bits 44:32).¹
- Bit 31 is always 0.
- Bits 44:32 report the number of bytes that software should allocate for the VMXON region and any VMCS region. It is a value greater than 0 and at most 4096 (bit 44 is set if and only if bits 43:32 are clear).
- Bit 48 indicates the width of the physical addresses that may be used for the VMXON region, each VMCS, and data structures referenced by pointers in a VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions). If the bit is 0, these addresses are limited to the processor's physical-address width.² If the bit is 1, these addresses are limited to 32 bits. This bit is always 0 for processors that support Intel 64 architecture.
- If bit 49 is read as 1, the logical processor supports the dual-monitor treatment of system-management interrupts and system-management mode. See Section 34.15 for details of this treatment.
- Bits 53:50 report the memory type that should be used for the VMCS, for data structures referenced by pointers in the VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions), and for the MSEG header. If software needs to access these data structures (e.g., to modify the contents of the MSR bitmaps), it can configure the paging structures to map them into the linear-address space. If it does so, it should establish mappings that use the memory type reported bits 53:50 in this MSR.³

As of this writing, all processors that support VMX operation indicate the write-back type. The values used are given in Table A-1.

Table A-1. Memory Types Recommended for VMCS and Related Data Structures

Value(s)	Field
0	Uncacheable (UC)
1-5	Not used
6	Write Back (WB)
7-15	Not used

1. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field in bits 31:0 of this MSR. For all processors produced prior to this change, bit 31 of this MSR was read as 0.
2. On processors that support Intel 64 architecture, the pointer must not set bits beyond the processor's physical address width. This width is denoted `MAXPHYADDR` and is derived from the value enumerated in `CPUID.80000008H:EAX[7:0]`. If `IA32_TME_ACTIVATE[0] = 1` (indicating that TME has been configured), `MAXPHYADDR` is reduced by the value of `IA32_TME_ACTIVATE[39:36]` when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35); the value is not reduced in SEAM.
3. Alternatively, software may map any of these regions or structures with the UC memory type. (This may be necessary for the MSEG header.) Doing so is discouraged unless necessary as it will cause the performance of software accesses to those structures to suffer.

- If bit 54 is read as 1, the processor reports information in the VM-exit instruction-information field on VM exits due to execution of the INS and OUTS instructions (see Section 30.2.5). This reporting is done only if this bit is read as 1.
- Bit 55 is read as 1 if any VMX controls that default to 1 may be cleared to 0. See Appendix A.2 for details. It also reports support for the VMX capability MSRs IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. See Appendix A.3.1, Appendix A.3.2, Appendix A.4, and Appendix A.5 for details.
- If bit 56 is read as 1, software can use VM entry to deliver a hardware exception with or without an error code, regardless of vector (see Section 29.2.1.3).
- If bit 58 is read as 1, the processor provides VMX nested-exception support. With this support, software may indicate that an injected event is a nested exception (see Section 29.2.1.3 and Section 29.6.1.1), and the VM exits will indicate whether an event is an exception encountered during FRED event delivery (see Section 30.2.2 and Section 30.2.4). Any processor that supports FRED transitions will provide VMX nested-exception support.
- The values of bits 47:45, bit 57, and bits 63:59 are reserved and are read as 0.

A.2 RESERVED CONTROLS AND DEFAULT SETTINGS

As noted in Chapter 29, “VM Entries,” certain VMX controls are reserved and must be set to a specific value (0 or 1) determined by the processor. The specific value to which a reserved control must be set is its **default setting**. Software can discover the default setting of a reserved control by consulting the appropriate VMX capability MSR (see Appendix A.3 through Appendix A.5).

Future processors may define new functionality for one or more reserved controls. Such processors would allow each newly defined control to be set either to 0 or to 1. Software that does not desire a control’s new functionality should set the control to its default setting. For that reason, it is useful for software to know the default settings of the reserved controls.

Default settings partition the various controls into the following classes:

- **Always-flexible.** These have never been reserved.
- **Default0.** These are (or have been) reserved with a default setting of 0.
- **Default1.** They are (or have been) reserved with a default setting of 1.

As noted in Appendix A.1, a logical processor uses bit 55 of the IA32_VMX_BASIC MSR to indicate whether any of the default1 controls may be 0:

- If bit 55 of the IA32_VMX_BASIC MSR is read as 0, all the default1 controls are reserved and must be 1. VM entry will fail if any of these controls are 0 (see Section 29.2.1).
- If bit 55 of the IA32_VMX_BASIC MSR is read as 1, not all the default1 controls are reserved, and some (but not necessarily all) may be 0. The CPU supports four (4) new VMX capability MSRs: IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. See Appendix A.3 through Appendix A.5 for details. (These MSRs are not supported if bit 55 of the IA32_VMX_BASIC MSR is read as 0.)

A.3 VM-EXECUTION CONTROLS

There are separate capability MSRs for the pin-based VM-execution controls, the primary processor-based VM-execution controls, the secondary processor-based VM-execution controls, and the tertiary processor-based VM-execution controls. These are described in Appendix A.3.1, Appendix A.3.2, Appendix A.3.3, and Appendix A.3.4, respectively.

A.3.1 Pin-Based VM-Execution Controls

The IA32_VMX_PINBASED_CTLMS MSR (index 481H) reports on the allowed settings of **most** of the pin-based VM-execution controls (see Section 27.6.1):

- Bits 31:0 indicate the **allowed 0-settings** of these controls. VM entry allows control X (bit X of the pin-based VM-execution controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the pin-based VM-execution controls in the default1 class (see Appendix A.2). These are bits 1, 2, and 4; the corresponding bits of the IA32_VMX_PINBASED_CTLMS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any pin-based VM-execution control in the default1 class is 0.
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLMS MSR (see below) reports which of the pin-based VM-execution controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the **allowed 1-settings** of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLMS MSR (index 48DH) reports on the allowed settings of **all** of the pin-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the pin-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_PINBASED_CTLMS MSR. (The IA32_VMX_TRUE_PINBASED_CTLMS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_TRUE_PINBASED_CTLMS MSR. Assuming that software knows that the default1 class of pin-based VM-execution controls contains bits 1, 2, and 4, there is no need for software to consult the IA32_VMX_PINBASED_CTLMS MSR.

A.3.2 Primary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLMS MSR (index 482H) reports on the allowed settings of **most** of the primary processor-based VM-execution controls (see Section 27.6.2):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the primary processor-based VM-execution controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the primary processor-based VM-execution controls in the default1 class (see Appendix A.2). These are bits 1, 4–6, 8, 13–16, and 26; the corresponding bits of the IA32_VMX_PROCBASED_CTLMS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any of the primary processor-based VM-execution controls in the default1 class is 0.
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PROCBASED_CTLMS MSR (see below) reports which of the primary processor-based VM-execution controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PROCBASED_CTLMSR (index 48EH) reports on the allowed settings of **all** of the primary processor-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the primary processor-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the primary processor-based VM-execution controls is contained in the IA32_VMX_PROCBASED_CTLMSR. (The IA32_VMX_TRUE_PROCBASED_CTLMSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the primary processor-based VM-execution controls is contained in the IA32_VMX_TRUE_PROCBASED_CTLMSR. Assuming that software knows that the default1 class of primary processor-based VM-execution controls contains bits 1, 4–6, 8, 13–16, and 26, there is no need for software to consult the IA32_VMX_PROCBASED_CTLMSR.

A.3.3 Secondary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLMSR2 (index 48BH) reports on the allowed settings of the secondary processor-based VM-execution controls (see Section 27.6.2). The following items provide details, including enforcement by VM entry:

- Bits 31:0 indicate the allowed 0-settings of these controls. These bits are always 0. This fact indicates that VM entry allows each bit of the secondary processor-based VM-execution controls to be 0 (reserved bits must be 0)
- Bits 63:32 indicate the allowed 1-settings of these controls; the 1-setting is not allowed for any reserved bit. VM entry allows control X (bit X of the secondary processor-based VM-execution controls) to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X and the “activate secondary controls” primary processor-based VM-execution control are both 1.

The IA32_VMX_PROCBASED_CTLMSR2 MSR exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLMSR is 1).

A.3.4 Tertiary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLMSR3 (index 492H) reports on the allowed 1-settings of the tertiary processor-based VM-execution controls (see Section 27.6.2); the 1-setting is not allowed for any reserved bit.

VM entry allows control X (bit X of the tertiary processor-based VM-execution controls) to be 1 if bit X in the MSR is set to 1; if bit X in the MSR is cleared to 0, VM entry fails if control X and the “activate tertiary controls” primary processor-based VM-execution control are both 1.

The IA32_VMX_PROCBASED_CTLMSR3 MSR exists only on processors that support the 1-setting of the “activate tertiary controls” VM-execution control (only if bit 49 of the IA32_VMX_PROCBASED_CTLMSR is 1).

Notice that the organization of this MSR differs from that of IA32_VMX_PROCBASED_CTLMSR2 (Appendix A.3.3). This is because there are 64 tertiary processor-based VM-execution controls, while there were only 32 secondary processor-based VM-execution controls.

A.4 VM-EXIT CONTROLS

There are separate capability MSRs for the primary VM-exit controls and the secondary VM-exit controls. These are described in Appendix A.4.1 and Appendix A.4.2, respectively.

A.4.1 Primary VM-Exit Controls

The `IA32_VMX_EXIT_CTLS` MSR (index 483H) reports on the allowed settings of **most** of the primary VM-exit controls (see Section 27.7.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the primary VM-exit controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the primary VM-exit controls in the default1 class (see Appendix A.2). These are bits 0–8, 10, 11, 13, 14, 16, and 17; the corresponding bits of the `IA32_VMX_EXIT_CTLS` MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the `IA32_VMX_BASIC` MSR:

- If bit 55 in the `IA32_VMX_BASIC` MSR is read as 0, VM entry fails if any primary VM-exit control in the default1 class is 0.
- If bit 55 in the `IA32_VMX_BASIC` MSR is read as 1, the `IA32_VMX_TRUE_EXIT_CTLS` MSR (see below) reports which of the primary VM-exit controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control 32+X to be 1 if bit X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the `IA32_VMX_BASIC` MSR is read as 1, the `IA32_VMX_TRUE_EXIT_CTLS` MSR (index 48FH) reports on the allowed settings of **all** of the primary VM-exit controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the primary VM-exit controls:

- If bit 55 in the `IA32_VMX_BASIC` MSR is read as 0, all information about the allowed settings of the primary VM-exit controls is contained in the `IA32_VMX_EXIT_CTLS` MSR. (The `IA32_VMX_TRUE_EXIT_CTLS` MSR is not supported.)
- If bit 55 in the `IA32_VMX_BASIC` MSR is read as 1, all information about the allowed settings of the primary VM-exit controls is contained in the `IA32_VMX_TRUE_EXIT_CTLS` MSR. Assuming that software knows that the default1 class of primary VM-exit controls contains bits 0–8, 10, 11, 13, 14, 16, and 17, there is no need for software to consult the `IA32_VMX_EXIT_CTLS` MSR.

A.4.2 Secondary VM-Exit Controls

The `IA32_VMX_EXIT_CTLS2` MSR (index 493H) reports on the allowed 1-settings of the secondary VM-exit controls (see Section 27.7.1); the 1-setting is not allowed for any reserved bit.

VM entry allows control X (bit X of the secondary VM-exit controls) to be 1 if bit X in the MSR is set to 1; if bit X in the MSR is cleared to 0, VM entry fails if control X and the “activate secondary controls” primary VM-exit control are both 1.

The `IA32_VMX_EXIT_CTLS2` MSR exists only on processors that support the 1-setting of the “activate secondary controls” VM-exit control (only if bit 63 of the `IA32_VMX_EXIT_CTLS` MSR is 1).

Notice that the organization of this MSR differs from that of `IA32_VMX_EXIT_CTLS` (Appendix A.4.1). This is because there are 64 secondary VM-exit controls, while there were only 32 primary VM-exit controls.

A.5 VM-ENTRY CONTROLS

The `IA32_VMX_ENTRY_CTLS` MSR (index 484H) reports on the allowed settings of **most** of the VM-entry controls (see Section 27.8.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the VM-entry controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the VM-entry controls in the default1 class (see Appendix A.2). These are bits 0–8 and 12; the corresponding bits of the IA32_VMX_ENTRY_CTLMSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any VM-entry control in the default1 class is 0.
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_ENTRY_CTLMSR (see below) reports which of the VM-entry controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the VM-entry controls and bit 32+X is 0 in this MSR.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_ENTRY_CTLMSR (index 490H) reports on the allowed settings of **all** of the VM-entry controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control 32+X to be 1 if bit X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the VM-entry controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the VM-entry controls is contained in the IA32_VMX_ENTRY_CTLMSR. (The IA32_VMX_TRUE_ENTRY_CTLMSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the VM-entry controls is contained in the IA32_VMX_TRUE_ENTRY_CTLMSR. Assuming that software knows that the default1 class of VM-entry controls contains bits 0–8 and 12, there is no need for software to consult the IA32_VMX_ENTRY_CTLMSR.

A.6 MISCELLANEOUS DATA

The IA32_VMX_MISC MSR (index 485H) consists of the following fields:

- Bits 4:0 report a value X that specifies the relationship between the rate of the VMX-preemption timer and that of the timestamp counter (TSC). Specifically, the VMX-preemption timer (if it is active) counts down by 1 every time bit X in the TSC changes due to a TSC increment.
- If bit 5 is read as 1, VM exits store the value of IA32_EFER.LMA into the “IA-32e mode guest” VM-entry control; see Section 30.2 for more details. This bit is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control.
- Bits 8:6 report, as a bitmap, the activity states supported by the implementation:
 - Bit 6 reports (if set) the support for activity state 1 (HLT).
 - Bit 7 reports (if set) the support for activity state 2 (shutdown).
 - Bit 8 reports (if set) the support for activity state 3 (wait-for-SIPI).

If an activity state is not supported, the implementation causes a VM entry to fail if it attempts to establish that activity state. All implementations support VM entry to activity state 0 (active).

- If bit 14 is read as 1, Intel® Processor Trace (Intel PT) can be used in VMX operation. If the processor supports Intel PT but does not allow it to be used in VMX operation, execution of VMXON clears IA32_RTIT_CTL.TraceEn (see “VMXON—Enter VMX Operation” in Chapter 33); any attempt to write IA32_RTIT_CTL while in VMX operation (including VMX root operation) causes a general-protection exception.
- If bit 15 is read as 1, the RDMSR instruction can be used in system-management mode (SMM) to read the IA32_SMBASE MSR (MSR address 9EH). See Section 34.15.6.3.
- Bits 24:16 indicate the number of CR3-target values supported by the processor. This number is a value between 0 and 256, inclusive (bit 24 is set if and only if bits 23:16 are clear).

- Bits 27:25 is used to compute the recommended maximum number of MSRs that should appear in the VM-exit MSR-store list, the VM-exit MSR-load list, or the VM-entry MSR-load list. Specifically, if the value bits 27:25 of `IA32_VMX_MISC` is N , then $512 * (N + 1)$ is the recommended maximum number of MSRs to be included in each list. If the limit is exceeded, undefined processor behavior may result (including a machine check during the VMX transition).
- If bit 28 is read as 1, bit 2 of the `IA32_SMM_MONITOR_CTL` can be set to 1. VMXOFF unblocks SMIs unless `IA32_SMM_MONITOR_CTL`[bit 2] is 1 (see Section 34.14.4).
- If bit 29 is read as 1, software can use VMWRITE to write to any supported field in the VMCS; otherwise, VMWRITE cannot be used to modify VM-exit information fields.
- If bit 30 is read as 1, VM entry allows injection of a software interrupt, software exception, or privileged software exception with an instruction length of 0.
- Bits 63:32 report the 32-bit MSEG revision identifier used by the processor.
- Bits 13:9 and bit 31 are reserved and are read as 0.

A.7 VMX-FIXED BITS IN CR0

The `IA32_VMX_CR0_FIXED0` MSR (index 486H) and `IA32_VMX_CR0_FIXED1` MSR (index 487H) indicate how bits in CR0 may be set in VMX operation. They report on bits in CR0 that are allowed to be 0 and to be 1, respectively, in VMX operation. If bit X is 1 in `IA32_VMX_CR0_FIXED0`, then that bit of CR0 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in `IA32_VMX_CR0_FIXED1`, then that bit of CR0 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in `IA32_VMX_CR0_FIXED0`, then that bit is also 1 in `IA32_VMX_CR0_FIXED1`; if bit X is 0 in `IA32_VMX_CR0_FIXED1`, then that bit is also 0 in `IA32_VMX_CR0_FIXED0`. Thus, each bit in CR0 is either fixed to 0 (with value 0 in both MSRs), fixed to 1 (1 in both MSRs), or flexible (0 in `IA32_VMX_CR0_FIXED0` and 1 in `IA32_VMX_CR0_FIXED1`).

A.8 VMX-FIXED BITS IN CR4

The `IA32_VMX_CR4_FIXED0` MSR (index 488H) and `IA32_VMX_CR4_FIXED1` MSR (index 489H) indicate how bits in CR4 may be set in VMX operation. They report on bits in CR4 that are allowed to be 0 and 1, respectively, in VMX operation. If bit X is 1 in `IA32_VMX_CR4_FIXED0`, then that bit of CR4 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in `IA32_VMX_CR4_FIXED1`, then that bit of CR4 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in `IA32_VMX_CR4_FIXED0`, then that bit is also 1 in `IA32_VMX_CR4_FIXED1`; if bit X is 0 in `IA32_VMX_CR4_FIXED1`, then that bit is also 0 in `IA32_VMX_CR4_FIXED0`. Thus, each bit in CR4 is either fixed to 0 (with value 0 in both MSRs), fixed to 1 (1 in both MSRs), or flexible (0 in `IA32_VMX_CR4_FIXED0` and 1 in `IA32_VMX_CR4_FIXED1`).

A.9 VMCS ENUMERATION

The `IA32_VMX_VMCS_ENUM` MSR (index 48AH) provides information to assist software in enumerating fields in the VMCS.

As noted in Section 27.11.2, each field in the VMCS is associated with a 32-bit encoding which is structured as follows:

- Bits 31:15 are reserved (must be 0).
- Bits 14:13 indicate the field's width.
- Bit 12 is reserved (must be 0).
- Bits 11:10 indicate the field's type.
- Bits 9:1 is an index field that distinguishes different fields with the same width and type.
- Bit 0 indicates access type.

IA32_VMX_VMCS_ENUM indicates to software the highest index value used in the encoding of any field supported by the processor:

- Bits 9:1 contain the highest index value used for any VMCS encoding.
- Bit 0 and bits 63:10 are reserved and are read as 0.

A.10 VPID AND EPT CAPABILITIES

The IA32_VMX_EPT_VPID_CAP MSR (index 48CH) reports information about the capabilities of the logical processor with regard to virtual-processor identifiers (VPIDs, Section 31.1) and extended page tables (EPT, Section 31.3):

- If bit 0 is read as 1, the processor supports execute-only translations by EPT. This support allows software to configure EPT paging-structure entries in which bits 1:0 are clear (indicating that data accesses are not allowed) and bit 2 is set (indicating that instruction fetches are allowed).¹
- Bit 6 indicates support for a page-walk length of 4.
- Bit 7 indicates support for a page-walk length of 5.
- If bit 8 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be uncacheable (UC); see Section 27.6.11.
- If bit 14 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be write-back (WB).
- If bit 16 is read as 1, the logical processor allows software to configure a EPT PDE to map a 2-Mbyte page (by setting bit 7 in the EPT PDE).
- If bit 17 is read as 1, the logical processor allows software to configure a EPT PDPTE to map a 1-Gbyte page (by setting bit 7 in the EPT PDPTE).
- Support for the INVEPT instruction (see Chapter 33 and Section 31.4.3.1):
 - If bit 20 is read as 1, the INVEPT instruction is supported.
 - If bit 25 is read as 1, the single-context INVEPT type is supported.
 - If bit 26 is read as 1, the all-context INVEPT type is supported.
- If bit 21 is read as 1, accessed and dirty flags for EPT are supported (see Section 31.3.5).
- If bit 22 is read as 1, the processor reports advanced VM-exit information for EPT violations (see Section 30.2.1). This reporting is done only if this bit is read as 1.
- If bit 23 is read as 1, supervisor shadow-stack control is supported (see Section 31.3.3.2).
- Support for the INVVPID instruction (see Chapter 33 and Section 31.4.3.1):
 - If bit 32 is read as 1, the INVVPID instruction is supported.
 - If bit 40 is read as 1, the individual-address INVVPID type is supported.
 - If bit 41 is read as 1, the single-context INVVPID type is supported.
 - If bit 42 is read as 1, the all-context INVVPID type is supported.
 - If bit 43 is read as 1, the single-context-retaining-globals INVVPID type is supported.
- Bits 53:48 enumerate the maximum HLAT prefix size. It is expected that any processor that supports the 1-setting of the “enable HLAT” VM-execution control will enumerate this value as 1. See Section 5.5.1.
- Bits 5:1, bits 13:9, bit 15, bits 19:18, bit 24, bits 31:27, bits 39:33, bits 47:44, and bits 63:54 are reserved and are read as 0.

The IA32_VMX_EPT_VPID_CAP MSR exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTL MSR is 1) and that support

1. If the “mode-based execute control for EPT” VM-execution control is 1, setting bit 0 indicates also that software may also configure EPT paging-structure entries in which bits 1:0 are both clear and in which bit 10 is set (indicating a translation that can be used to fetch instructions from a supervisor-mode linear address or a user-mode linear address).

either the 1-setting of the “enable EPT” VM-execution control (only if bit 33 of the IA32_VMX_PROCBASED_CTLSS2 MSR is 1) or the 1-setting of the “enable VPID” VM-execution control (only if bit 37 of the IA32_VMX_PROCBASED_CTLSS2 MSR is 1).

A.11 VM FUNCTIONS

The IA32_VMX_VMFUNC MSR (index 491H) reports on the allowed settings of the VM-function controls (see Section 27.6.14). VM entry allows bit X of the VM-function controls to be 1 if bit X in the MSR is set to 1; if bit X in the MSR is cleared to 0, VM entry fails if bit X of the VM-function controls, the “activate secondary controls” primary processor-based VM-execution control, and the “enable VM functions” secondary processor-based VM-execution control are all 1.

The IA32_VMX_VMFUNC MSR exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLSS MSR is 1) and the 1-setting of the “enable VM functions” secondary processor-based VM-execution control (only if bit 45 of the IA32_VMX_PROCBASED_CTLSS2 MSR is 1).

31. Updates Appendix B, Volume 3D

Change bars and text show changes to Appendix B of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- In Table B-4. "Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)", added "Injected-event data" fields for processors that support FRED transitions.
- In Table B-5. "Encodings for 64-Bit Read-Only Data Fields (0010_01xx_xxxx_xxxAb)", added "Original-event data" fields for processors that support FRED transitions.
- In Table B-6. "Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)", added Guest IA32_FRED_* fields for processors that support "load FRED" VM-entry control or "save FRED" VM-exit control.
- In Table B-7. "Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb)", added Host IA32_FRED_* fields for processors that support "load FRED" VM-exit control.
- In Table B-8. "Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B)", updated field names for VM-entry with Injected-event.
- In Table B-9. "Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B)", updated field names replacing VM-exit with Exiting-event and replacing IDT-vectoring with Original event.

APPENDIX B FIELD ENCODING IN VMCS

Every component of the VMCS is encoded by a 32-bit field that can be used by VMREAD and VMWRITE. Section 27.11.2 describes the structure of the encoding space (the meanings of the bits in each 32-bit encoding).

This appendix enumerates all fields in the VMCS and their encodings. Fields are grouped by width (16-bit, 32-bit, etc.) and type (guest-state, host-state, etc.).

B.1 16-BIT FIELDS

A value of 0 in bits 14:13 of an encoding indicates a 16-bit field. Only guest-state areas and the host-state area contain 16-bit fields. As noted in Section 27.11.2, each 16-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.1.1 16-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-1 enumerates the 16-bit control fields.

Table B-1. Encoding for 16-Bit Control Fields (0000_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Virtual-processor identifier (VPID) ¹	000000000B	00000000H
Posted-interrupt notification vector ²	000000001B	00000002H
EPTP index ³	000000010B	00000004H
HLAT prefix size ⁴	000000011B	00000006H
Last PID-pointer index ⁵	000000100B	00000008H

NOTES:

1. This field exists only on processors that support the 1-setting of the “enable VPID” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “process posted interrupts” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “enable HLAT” VM-execution control.
5. This field exists only on processors that support the 1-setting of the “IPI virtualization” VM-execution control.

B.1.2 16-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-2 enumerates 16-bit guest-state fields.

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES selector	000000000B	00000800H
Guest CS selector	000000001B	00000802H
Guest SS selector	000000010B	00000804H
Guest DS selector	000000011B	00000806H
Guest FS selector	000000100B	00000808H

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest GS selector	000000101B	0000080AH
Guest LDTR selector	000000110B	0000080CH
Guest TR selector	000000111B	0000080EH
Guest interrupt status ¹	000001000B	00000810H
PML index ²	000001001B	00000812H
Guest UINV ³	000001010B	00000814H

NOTES:

1. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable PML” VM-execution control.
3. This field exists only on processors that support the 1-setting of either the “clear UINV” VM-exit control or the “load UINV” VM-entry control.

B.1.3 16-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-3 enumerates the 16-bit host-state fields.

Table B-3. Encodings for 16-Bit Host-State Fields (0000_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host ES selector	000000000B	00000C00H
Host CS selector	000000001B	00000C02H
Host SS selector	000000010B	00000C04H
Host DS selector	000000011B	00000C06H
Host FS selector	000000100B	00000C08H
Host GS selector	000000101B	00000C0AH
Host TR selector	000000110B	00000C0CH

B.2 64-BIT FIELDS

A value of 1 in bits 14:13 of an encoding indicates a 64-bit field. There are 64-bit fields only for controls and for guest state. As noted in Section 27.11.2, every 64-bit field has two encodings, which differ on bit 0, the access type. Thus, each such field has an even encoding for full access and an odd encoding for high access.

B.2.1 64-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-4 enumerates the 64-bit control fields.

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)

Field Name	Index	Encoding
Address of I/O bitmap A (full)	000000000B	00002000H
Address of I/O bitmap A (high)		00002001H
Address of I/O bitmap B (full)	000000001B	00002002H
Address of I/O bitmap B (high)		00002003H

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Address of MSR bitmaps (full) ¹	000000010B	00002004H
Address of MSR bitmaps (high) ¹		00002005H
VM-exit MSR-store address (full)	000000011B	00002006H
VM-exit MSR-store address (high)		00002007H
VM-exit MSR-load address (full)	000000100B	00002008H
VM-exit MSR-load address (high)		00002009H
VM-entry MSR-load address (full)	000000101B	0000200AH
VM-entry MSR-load address (high)		0000200BH
Executive-VMCS pointer (full)	000000110B	0000200CH
Executive-VMCS pointer (high)		0000200DH
PML address (full) ²	000000111B	0000200EH
PML address (high) ²		0000200FH
TSC offset (full)	000001000B	00002010H
TSC offset (high)		00002011H
Virtual-APIC address (full) ³	000001001B	00002012H
Virtual-APIC address (high) ³		00002013H
APIC-access address (full) ⁴	000001010B	00002014H
APIC-access address (high) ⁴		00002015H
Posted-interrupt descriptor address (full) ⁵	000001011B	00002016H
Posted-interrupt descriptor address (high) ⁵		00002017H
VM-function controls (full) ⁶	000001100B	00002018H
VM-function controls (high) ⁶		00002019H
EPT pointer (EPTP; full) ⁷	000001101B	0000201AH
EPT pointer (EPTP; high) ⁷		0000201BH
EOI-exit bitmap 0 (EOI_EXIT0; full) ⁸	000001110B	0000201CH
EOI-exit bitmap 0 (EOI_EXIT0; high) ⁸		0000201DH
EOI-exit bitmap 1 (EOI_EXIT1; full) ⁸	000001111B	0000201EH
EOI-exit bitmap 1 (EOI_EXIT1; high) ⁸		0000201FH
EOI-exit bitmap 2 (EOI_EXIT2; full) ⁸	000010000B	00002020H
EOI-exit bitmap 2 (EOI_EXIT2; high) ⁸		00002021H
EOI-exit bitmap 3 (EOI_EXIT3; full) ⁸	000010001B	00002022H
EOI-exit bitmap 3 (EOI_EXIT3; high) ⁸		00002023H
EPTP-list address (full) ⁹	000010010B	00002024H
EPTP-list address (high) ⁹		00002025H
VMREAD-bitmap address (full) ¹⁰	000010011B	00002026H
VMREAD-bitmap address (high) ¹⁰		00002027H
VMWRITE-bitmap address (full) ¹⁰	000010100B	00002028H
VMWRITE-bitmap address (high) ¹⁰		00002029H

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Virtualization-exception information address (full) ¹¹	000010101B	0000202AH
Virtualization-exception information address (high) ¹¹		0000202BH
XSS-exiting bitmap (full) ¹²	000010110B	0000202CH
XSS-exiting bitmap (high) ¹²		0000202DH
ENCLS-exiting bitmap (full) ¹³	000010111B	0000202EH
ENCLS-exiting bitmap (high) ¹³		0000202FH
Sub-page-permission-table pointer (full) ¹⁴	000011000B	00002030H
Sub-page-permission-table pointer (high) ¹⁴		00002031H
TSC multiplier (full) ¹⁵	000011001B	00002032H
TSC multiplier (high) ¹⁵		00002033H
Tertiary processor-based VM-execution controls (full) ¹⁶	000011010B	00002034H
Tertiary processor-based VM-execution controls (high) ¹⁶		00002035H
Low PASID directory address (full) ¹⁷	000011100B	00002038H
Low PASID directory address (high) ¹⁷		00002039H
High PASID directory address (full) ¹⁷	000011101B	0000203AH
High PASID directory address (high) ¹⁷		0000203BH
SEAM shared EPT pointer (full) ¹⁸	000011110B	0000203CH
SEAM shared EPT pointer (high) ¹⁸		0000203DH
PCONFIG-exiting bitmap (full) ¹⁹	000011111B	0000203EH
PCONFIG-exiting bitmap (high) ¹⁹		0000203FH
Hypervisor-managed linear-address translation pointer (HLATP; full) ²⁰	000100000B	00002040H
HLATP (high) ²⁰		00002041H
PID-pointer table address (full) ²¹	000100001B	00002042H
PID-pointer table address (high) ²¹		00002043H
Secondary VM-exit controls (full) ²²	000100010B	00002044H
Secondary VM-exit controls (high) ²²		00002045H
IA32_SPEC_CTRL mask (full) ²³	000100101B	0000204AH
IA32_SPEC_CTRL mask (high) ²³		0000204BH
IA32_SPEC_CTRL shadow (full) ²³	000100110B	0000204CH
IA32_SPEC_CTRL shadow (high) ²³		0000204DH
Injected-event data (full) ²⁴	000101001B	00002052H
Injected-event data (high) ²⁴		00002053H

NOTES:

1. This field exists only on processors that support the 1-setting of the “use MSR bitmaps” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable PML” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “virtualize APIC accesses” VM-execution control.
5. This field exists only on processors that support the 1-setting of the “process posted interrupts” VM-execution control.
6. This field exists only on processors that support the 1-setting of the “enable VM functions” VM-execution control.
7. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.

8. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.
9. This field exists only on processors that support the 1-setting of the “EPTP switching” VM-function control.
10. This field exists only on processors that support the 1-setting of the “VMCS shadowing” VM-execution control.
11. This field exists only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.
12. This field exists only on processors that support the 1-setting of the “enable XSAVES/XRSTORS” VM-execution control.
13. This field exists only on processors that support the 1-setting of the “enable ENCLS exiting” VM-execution control.
14. This field exists only on processors that support the 1-setting of the “sub-page write permissions for EPT” VM-execution control.
15. This field exists only on processors that support the 1-setting of the “use TSC scaling” VM-execution control.
16. This field exists only on processors that support the 1-setting of the “activate tertiary controls” VM-execution control.
17. This field exists only on processors that support the 1-setting of the “PASID translation” VM-execution control.
18. This field exists only on processors that support the 1-setting of the “SEAM guest-physical address width” VM-execution control.
19. This field exists only on processors that support the 1-setting of the “enable PCONFIG” VM-execution control.
20. This field exists only on processors that support the 1-setting of the “enable HLAT” VM-execution control.
21. This field exists only on processors that support the 1-setting of the “IPI virtualization” VM-execution control.
22. This field exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control.
23. This field exists only on processors that support the 1-setting of the “virtualize IA32_SPEC_CTRL” VM-execution control.
24. This field exists only on processors that support FRED transitions.

B.2.2 64-Bit Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-5 enumerates the 64-bit read-only data fields.

Table B-5. Encodings for 64-Bit Read-Only Data Fields (0010_01xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest-physical address (full) ¹	000000000B	00002400H
Guest-physical address (high) ¹		00002401H
MSR data (full) ²	000000001B	00002402H
MSR data (high) ²		00002403H
Original-event data (full) ³	000000010B	00002404H
Original-event data (high) ³		00002405H

NOTES:

1. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable MSR-list instructions” VM-execution control.
3. This field exists only on processors that support FRED transitions.

B.2.3 64-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-6 enumerates the 64-bit guest-state fields.

Table B-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)

Field Name	Index	Encoding
VMCS link pointer (full)	000000000B	00002800H
VMCS link pointer (high)		00002801H
Guest IA32_DEBUGCTL (full)	000000001B	00002802H
Guest IA32_DEBUGCTL (high)		00002803H

Table B-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Guest IA32_PAT (full) ¹	000000010B	00002804H
Guest IA32_PAT (high) ¹		00002805H
Guest IA32_EFER (full) ²	000000011B	00002806H
Guest IA32_EFER (high) ²		00002807H
Guest IA32_PERF_GLOBAL_CTRL (full) ³	000000100B	00002808H
Guest IA32_PERF_GLOBAL_CTRL (high) ³		00002809H
Guest PDPTE0 (full) ⁴	000000101B	0000280AH
Guest PDPTE0 (high) ⁴		0000280BH
Guest PDPTE1 (full) ⁴	000000110B	0000280CH
Guest PDPTE1 (high) ⁴		0000280DH
Guest PDPTE2 (full) ⁴	000000111B	0000280EH
Guest PDPTE2 (high) ⁴		0000280FH
Guest PDPTE3 (full) ⁴	000001000B	00002810H
Guest PDPTE3 (high) ⁴		00002811H
Guest IA32_BNDCFGS (full) ⁵	000001001B	00002812H
Guest IA32_BNDCFGS (high) ⁵		00002813H
Guest IA32_RTIT_CTL (full) ⁶	000001010B	00002814H
Guest IA32_RTIT_CTL (high) ⁶		00002815H
Guest IA32_LBR_CTL (full) ⁷	000001011B	00002816H
Guest IA32_LBR_CTL (high) ⁷		00002817H
Guest IA32_PKRS (full) ⁸	000001100B	00002818H
Guest IA32_PKRS (high) ⁸		00002819H
Guest IA32_FRED_CONFIG (full) ⁹	000001101B	0000281AH
Guest IA32_FRED_CONFIG (high) ⁹		0000281BH
Guest IA32_FRED_RSP1 (full) ⁹	000001110B	0000281CH
Guest IA32_FRED_RSP1 (high) ⁹		0000281DH
Guest IA32_FRED_RSP2 (full) ⁹	000001111B	0000281EH
Guest IA32_FRED_RSP2 (high) ⁹		0000281FH
Guest IA32_FRED_RSP3 (full) ⁹	000010000B	00002820H
Guest IA32_FRED_RSP3 (high) ⁹		00002821H
Guest IA32_FRED_STKLVLS (full) ⁹	000010001B	00002822H
Guest IA32_FRED_STKLVLS (high) ⁹		00002823H
Guest IA32_FRED_SSP1 (full) ⁹	000010010B	00002824H
Guest IA32_FRED_SSP1 (high) ⁹		00002825H
Guest IA32_FRED_SSP2 (full) ⁹	000010011B	00002826H
Guest IA32_FRED_SSP2 (high) ⁹		00002827H
Guest IA32_FRED_SSP3 (full) ⁹	000010100B	00002828H
Guest IA32_FRED_SSP3 (high) ⁹		00002829H

NOTES:

1. This field exists only on processors that support either the 1-setting of the “load IA32_PAT” VM-entry control or that of the “save IA32_PAT” VM-exit control.
2. This field exists only on processors that support either the 1-setting of the “load IA32_EFER” VM-entry control or that of the “save IA32_EFER” VM-exit control.
3. This field exists only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-entry control.
4. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.
5. This field exists only on processors that support either the 1-setting of the “load IA32_BNDCFGS” VM-entry control or that of the “clear IA32_BNDCFGS” VM-exit control.
6. This field exists only on processors that support either the 1-setting of the “load IA32_RTIT_CTL” VM-entry control or that of the “clear IA32_RTIT_CTL” VM-exit control.
7. This field exists only on processors that support either the 1-setting of the “load IA32_LBR_CTL” VM-entry control or that of the “clear IA32_LBR_CTL” VM-exit control.
8. This field exists only on processors that support the 1-setting of the “load PKRS” VM-entry control.
9. This field exists only on processors that support either the 1-setting of the “load FRED” VM-entry control or that of the “save FRED” VM-exit control.

B.2.4 64-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-7 enumerates the 64-bit control fields.

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb)

Field Name	Index	Encoding
Host IA32_PAT (full) ¹	000000000B	00002C00H
Host IA32_PAT (high) ¹		00002C01H
Host IA32_EFER (full) ²	000000001B	00002C02H
Host IA32_EFER (high) ²		00002C03H
Host IA32_PERF_GLOBAL_CTRL (full) ³	000000010B	00002C04H
Host IA32_PERF_GLOBAL_CTRL (high) ³		00002C05H
Host IA32_PKRS (full) ⁴	000000011B	00002C06H
Host IA32_PKRS (high) ⁴		00002C07H
Host IA32_FRED_CONFIG (full) ⁵	000000100B	00002C08H
Host IA32_FRED_CONFIG (high) ⁴		00002C09H
Host IA32_FRED_RSP1 (full) ⁴	000000101B	00002C0AH
Host IA32_FRED_RSP1 (high) ⁴		00002C0BH
Host IA32_FRED_RSP2 (full) ⁴	000000110B	00002C0CH
Host IA32_FRED_RSP2 (high) ⁴		00002C0DH
Host IA32_FRED_RSP3 (full) ⁴	000000111B	00002C0EH
Host IA32_FRED_RSP3 (high) ⁴		00002C0FH
Host IA32_FRED_STKLVLS (full) ⁴	000001000B	00002C10H
Host IA32_FRED_STKLVLS (high) ⁴		00002C11H
Host IA32_FRED_SSP1 (full) ⁴	000001001B	00002C12H
Host IA32_FRED_SSP1 (high) ⁴		00002C13H

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Host IA32_FRED_SSP2 (full) ⁴	000001010B	00002C14H
Host IA32_FRED_SSP2 (high) ⁴		00002C15H
Host IA32_FRED_SSP3 (full) ⁴	000001011B	00002C16H
Host IA32_FRED_SSP3 (high) ⁴		00002C17H

NOTES:

1. This field exists only on processors that support the 1-setting of the “load IA32_PAT” VM-exit control.
2. This field exists only on processors that support the 1-setting of the “load IA32_EFER” VM-exit control.
3. This field exists only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-exit control.
4. This field exists only on processors that support the 1-setting of the “load PKRS” VM-exit control.
5. This field exists only on processors that support the 1-setting of the “load FRED” VM-exit control.

B.3 32-BIT FIELDS

A value of 2 in bits 14:13 of an encoding indicates a 32-bit field. As noted in Section 27.11.2, each 32-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.3.1 32-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-8 enumerates the 32-bit control fields.

Table B-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Pin-based VM-execution controls	000000000B	00004000H
Primary processor-based VM-execution controls	000000001B	00004002H
Exception bitmap	000000010B	00004004H
Page-fault error-code mask	000000011B	00004006H
Page-fault error-code match	000000100B	00004008H
CR3-target count	000000101B	0000400AH
Primary VM-exit controls	000000110B	0000400CH
VM-exit MSR-store count	000000111B	0000400EH
VM-exit MSR-load count	000001000B	00004010H
VM-entry controls	000001001B	00004012H
VM-entry MSR-load count	000001010B	00004014H
Injected-event identification ¹	000001011B	00004016H
Injected-event error code ²	000001100B	00004018H
VM-entry instruction length	000001101B	0000401AH
TPR threshold ³	000001110B	0000401CH
Secondary processor-based VM-execution controls ⁴	000001111B	0000401EH
PLE_Gap ⁵	000010000B	00004020H
PLE_Window ⁵	000010001B	00004022H

Table B-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Instruction-timeout control ⁶	000010010B	00004024H
SEAM-guest KeyID ⁷	000010011B	00004026H

NOTES:

1. Older versions of this document called this field VM-entry interruption information.
2. Older versions of this document called this field VM-entry error code.
3. This field exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control.
5. This field exists only on processors that support the 1-setting of the “PAUSE-loop exiting” VM-execution control.
6. This field exists only on processors that support the 1-setting of the “instruction timeout” VM-execution control.
7. This field exists only on processors that support the 1-setting of the “SEAM guest-physical address width” VM-execution control.

B.3.2 32-Bit Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-9 enumerates the 32-bit read-only data fields.

Table B-9. Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
VM-instruction error	000000000B	00004400H
Exit reason	000000001B	00004402H
Exiting-event identification ¹	000000010B	00004404H
Exiting-event error code ²	000000011B	00004406H
Original-event identification ³	000000100B	00004408H
Original-event error code ⁴	000000101B	0000440AH
VM-exit instruction length	000000110B	0000440CH
VM-exit instruction information	000000111B	0000440EH

NOTES:

1. Older versions of this document called this field VM-exit interruption information.
2. Older versions of this document called this field VM-exit interruption error code.
3. Older versions of this document called this field IDT-vectoring information.
4. Older versions of this document called this field IDT-vectoring error code.

B.3.3 32-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-10 enumerates the 32-bit guest-state fields.

Table B-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES limit	000000000B	00004800H
Guest CS limit	000000001B	00004802H
Guest SS limit	000000010B	00004804H
Guest DS limit	000000011B	00004806H

Table B-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest FS limit	000000100B	00004808H
Guest GS limit	000000101B	0000480AH
Guest LDTR limit	000000110B	0000480CH
Guest TR limit	000000111B	0000480EH
Guest GDTR limit	000001000B	00004810H
Guest IDTR limit	000001001B	00004812H
Guest ES access rights	000001010B	00004814H
Guest CS access rights	000001011B	00004816H
Guest SS access rights	000001100B	00004818H
Guest DS access rights	000001101B	0000481AH
Guest FS access rights	000001110B	0000481CH
Guest GS access rights	000001111B	0000481EH
Guest LDTR access rights	000010000B	00004820H
Guest TR access rights	000010001B	00004822H
Guest interruptibility state	000010010B	00004824H
Guest activity state	000010011B	00004826H
Guest SMBASE	000010100B	00004828H
Guest IA32_SYSENTER_CS	000010101B	0000482AH
VMX-preemption timer value ¹	000010111B	0000482EH

NOTES:

1. This field exists only on processors that support the 1-setting of the “activate VMX-preemption timer” VM-execution control.

The limit fields for GDTR and IDTR are defined to be 32 bits in width even though these fields are only 16-bits wide in the Intel 64 and IA-32 architectures. VM entry ensures that the high 16 bits of both these fields are cleared to 0.

B.3.4 32-Bit Host-State Field

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. There is only one such 32-bit field as given in Table B-11.

Table B-11. Encoding for 32-Bit Host-State Field (0100_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host IA32_SYSENTER_CS	000000000B	00004C00H

B.4 NATURAL-WIDTH FIELDS

A value of 3 in bits 14:13 of an encoding indicates a natural-width field. As noted in Section 27.11.2, each of these fields allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.4.1 Natural-Width Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-12 enumerates the natural-width control fields.

Table B-12. Encodings for Natural-Width Control Fields (0110_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
CR0 guest/host mask	000000000B	00006000H
CR4 guest/host mask	000000001B	00006002H
CR0 read shadow	000000010B	00006004H
CR4 read shadow	000000011B	00006006H
CR3-target value 0	000000100B	00006008H
CR3-target value 1	000000101B	0000600AH
CR3-target value 2	000000110B	0000600CH
CR3-target value 3 ¹	000000111B	0000600EH

NOTES:

1. If a future implementation supports more than 4 CR3-target values, they will be encoded consecutively following the 4 encodings given here.

B.4.2 Natural-Width Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-13 enumerates the natural-width read-only data fields.

Table B-13. Encodings for Natural-Width Read-Only Data Fields (0110_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
Exit qualification	000000000B	00006400H
I/O RCX	000000001B	00006402H
I/O RSI	000000010B	00006404H
I/O RDI	000000011B	00006406H
I/O RIP	000000100B	00006408H
Guest-linear address	000000101B	0000640AH

B.4.3 Natural-Width Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-14 enumerates the natural-width guest-state fields.

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest CR0	000000000B	00006800H
Guest CR3	000000001B	00006802H
Guest CR4	000000010B	00006804H
Guest ES base	000000011B	00006806H
Guest CS base	000000100B	00006808H
Guest SS base	000000101B	0000680AH
Guest DS base	000000110B	0000680CH
Guest FS base	000000111B	0000680EH

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest GS base	000001000B	00006810H
Guest LDTR base	000001001B	00006812H
Guest TR base	000001010B	00006814H
Guest GDTR base	000001011B	00006816H
Guest IDTR base	000001100B	00006818H
Guest DR7	000001101B	0000681AH
Guest RSP	000001110B	0000681CH
Guest RIP	000001111B	0000681EH
Guest RFLAGS	000010000B	00006820H
Guest pending debug exceptions	000010001B	00006822H
Guest IA32_SYSENTER_ESP	000010010B	00006824H
Guest IA32_SYSENTER_EIP	000010011B	00006826H
Guest IA32_S_CET ¹	000010100B	00006828H
Guest SSP ¹	000010101B	0000682AH
Guest IA32_INTERRUPT_SSP_TABLE_ADDR ¹	000010110B	0000682CH

NOTES:

1. This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.

The base-address fields for ES, CS, SS, and DS in the guest-state area are defined to be natural-width (with 64 bits on processors supporting Intel 64 architecture) even though these fields are only 32-bits wide in the Intel 64 architecture. VM entry ensures that the high 32 bits of these fields are cleared to 0.

B.4.4 Natural-Width Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-15 enumerates the natural-width host-state fields.

Table B-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host CR0	000000000B	00006C00H
Host CR3	000000001B	00006C02H
Host CR4	000000010B	00006C04H
Host FS base	000000011B	00006C06H
Host GS base	000000100B	00006C08H
Host TR base	000000101B	00006C0AH
Host GDTR base	000000110B	00006C0CH
Host IDTR base	000000111B	00006C0EH
Host IA32_SYSENTER_ESP	000001000B	00006C10H
Host IA32_SYSENTER_EIP	000001001B	00006C12H
Host RSP	000001010B	00006C14H
Host RIP	000001011B	00006C16H

Table B-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Host IA32_S_CET ¹	000001100B	00006C18H
Host SSP ¹	000001101B	00006C1AH
Host IA32_INTERRUPT_SSP_TABLE_ADDR ¹	000001110B	00006C1CH

NOTES:

1. This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.

32. Updates to Chapter 2, Volume 4

Change bars and violet text show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers*.

Changes to this chapter:

- In Table 2-2, "IA-32 Architectural MSRs," updated IA32_PL0_SSP for Flexible Return and Event Delivery (FRED).
- In Table 2-2, "IA-32 Architectural MSRs," added the following MSRs for FRED:
 - FRED RSP MSRs:
 - IA32_FRED_RSP0 (1CCH)
 - IA32_FRED_RSP1 (1CDH)
 - IA32_FRED_RSP2 (1CEH)
 - IA32_FRED_RSP3 (1CFH)
 - IA32_FRED_STKLVLS (1D0H)
 - FRED SSP MSRs:
 - IA32_FRED_SSP1 (1D1H)
 - IA32_FRED_SSP2 (1D2H)
 - IA32_FRED_SSP3 (1D3H)
 - IA32_FRED_CONFIG (1D4H)
- In Table 2-2, "IA-32 Architectural MSRs," added the following MSRs for Architectural PEBS:
 - IA32_PEBS_BASE (3F4H)
 - IA32_PEBS_INDEX (3F5H)
 - IA32_PMC_GP0_CFG_C (1903H)
 - IA32_PMC_GP8_CFG_B (1922H)
 - IA32_PMC_GP8_CFG_C (1923H)
 - IA32_PMC_GP9_CFG_B (1926H)
 - IA32_PMC_GP9_CFG_C (1927H)
 - IA32_PMC_FX0_CFG_C (1983H)
 - IA32_PMC_FX2_CFG_B (198AH)
 - IA32_PMC_FX3_CFG_B (198EH)
 - IA32_PMC_FX3_CFG_C (198FH)
- In Table 2-2, "IA-32 Architectural MSRs," added the IA32_TSE_CAPABILITY MSR (9F1H) for Total Storage Encryption.
- In Table 2-39, updated MSR_PLATFORM_POWER_LIMIT (65CH) for 4th and 5th generation Intel® Xeon® Scalable Processor Families, Intel® Xeon® 6 E-Core Processors, and Intel® Xeon 6 P-core Processors increasing the bit field width for the Power Platform Limit field by two bits.

CHAPTER 2

MODEL-SPECIFIC REGISTERS (MSRS)

This chapter lists MSRs across Intel processor families. All MSRs listed can be read with the RDMSR and written with the WRMSR instructions. The scope of an MSR defines the set of processors that access the same MSR with RDMSR and WRMSR. Thread-scope MSRs are unique to every logical processor. Core-scope MSRs are shared by the threads in the same core; similarly for module-scope, die-scope, and package-scope.

When a processor package contains a single die, die-scope and package-scope are synonymous. When a package contains multiple die, they are distinct.

NOTE

For information on hierarchical level types supported, refer to the CPUID.1FH definition for the actual level type numbers: "V2 Extended Topology Enumeration Leaf" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A. Also see Section 11.9.1, "Hierarchical Mapping of Shared Resources," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

Register addresses are given in both hexadecimal and decimal. The register name is the mnemonic register name and the bit description describes individual bits in registers.

Model specific registers and its bit-fields may be supported for a finite range of processor families/models. To distinguish between different processor family and/or models, software must use CPUID.01H to query the combination of DisplayFamily and DisplayModel to determine model-specific availability of MSRs (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A). Table 2-1 lists the signature values of DisplayFamily and DisplayModel for various processor families or processor number series.

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_BDH	Intel® Series 2 Core™ Ultra processors supporting Lunar Lake performance hybrid architecture
06_ADH, 06_AEH	Intel® Xeon® 6 P-core processors based on Granite Rapids microarchitecture
06_AFH	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture
06_AAH	Intel® Core™ Ultra 7 processors supporting Meteor Lake performance hybrid architecture
06_CFH	5th generation Intel® Xeon® Scalable Processor Family based on Emerald Rapids microarchitecture
06_8FH	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture
06_BAH, 06_B7H, 06_BFH	13th generation Intel® Core™ processors supporting Raptor Lake performance hybrid architecture
06_97H, 06_9AH	12th generation Intel® Core™ processors supporting Alder Lake performance hybrid architecture
06_8CH, 06_8DH	11th generation Intel® Core™ processors based on Tiger Lake microarchitecture
06_A7H	11th generation Intel® Core™ processors based on Rocket Lake microarchitecture
06_7EH	10th generation Intel® Core™ processors based on Ice Lake microarchitecture
06_A5H, 06_A6H	10th generation Intel® Core™ processors based on Comet Lake microarchitecture
06_66H	Intel® Core™ processors based on Cannon Lake microarchitecture
06_8EH, 06_9EH	7th generation Intel® Core™ processors based on Kaby Lake microarchitecture, 8th and 9th generation Intel® Core™ processors based on Coffee Lake microarchitecture, Intel® Xeon® E processors based on Coffee Lake microarchitecture
06_6AH, 06_6CH	3rd generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_55H	Intel® Xeon® Scalable Processor Family based on Skylake microarchitecture, 2nd generation Intel® Xeon® Scalable Processor Family based on Cascade Lake product, and 3rd generation Intel® Xeon® Scalable Processor Family based on Cooper Lake product
06_4EH, 06_5EH	6th generation Intel Core processors and Intel Xeon processor E3-1500m v5 product family and E3-1200 v5 product family based on Skylake microarchitecture
06_85H	Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series based on Knights Mill microarchitecture
06_57H	Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series based on Knights Landing microarchitecture
06_56H	Intel Xeon processor D-1500 product family based on Broadwell microarchitecture
06_4FH	Intel Xeon processor E5 v4 Family based on Broadwell microarchitecture, Intel Xeon processor E7 v4 Family, Intel Core i7-69xx Processor Extreme Edition
06_47H	5th generation Intel Core processors, Intel Xeon processor E3-1200 v4 product family based on Broadwell microarchitecture
06_3DH	Intel Core M-5xxx Processor, 5th generation Intel Core processors based on Broadwell microarchitecture
06_3FH	Intel Xeon processor E5-4600/2600/1600 v3 product families, Intel Xeon processor E7 v3 product families based on Haswell-E microarchitecture, Intel Core i7-59xx Processor Extreme Edition
06_3CH, 06_45H, 06_46H	4th Generation Intel Core processor and Intel Xeon processor E3-1200 v3 product family based on Haswell microarchitecture
06_3EH	Intel Xeon processor E7-8800/4800/2800 v2 product families based on Ivy Bridge-E microarchitecture
06_3EH	Intel Xeon processor E5-2600/1600 v2 product families and Intel Xeon processor E5-2400 v2 product family based on Ivy Bridge-E microarchitecture, Intel Core i7-49xx Processor Extreme Edition
06_3AH	3rd Generation Intel Core Processor and Intel Xeon processor E3-1200 v2 product family based on Ivy Bridge microarchitecture
06_2DH	Intel Xeon processor E5 Family based on Sandy Bridge microarchitecture, Intel Core i7-39xx Processor Extreme Edition
06_2FH	Intel Xeon Processor E7 Family
06_2AH	Intel Xeon processor E3-1200 product family; 2nd Generation Intel Core i7, i5, i3 Processors 2xxx Series
06_2EH	Intel Xeon processor 7500, 6500 series
06_25H, 06_2CH	Intel Xeon processors 3600, 5600 series, Intel Core i7, i5, and i3 Processors
06_1EH, 06_1FH	Intel Core i7 and i5 Processors
06_1AH	Intel Core i7 Processor, Intel Xeon processor 3400, 3500, 5500 series
06_1DH	Intel Xeon processor MP 7400 series
06_17H	Intel Xeon processor 3100, 3300, 5200, 5400 series, Intel Core 2 Quad processors 8000, 9000 series
06_0FH	Intel Xeon processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad processor 6000 series, Intel Core 2 Extreme 6000 series, Intel Core 2 Duo 4000, 5000, 6000, 7000 series processors, Intel Pentium dual-core processors
06_0EH	Intel Core Duo, Intel Core Solo processors
06_0DH	Intel Pentium M processor
06_86H, 06_96H, 06_9CH	Intel Atom® processors, Intel® Celeron® processors, Intel® Pentium® processors, and Intel® Pentium® Silver processors based on Tremont Microarchitecture
06_7AH	Intel Atom processors based on Goldmont Plus microarchitecture
06_5FH	Intel Atom processors based on Goldmont microarchitecture (Denverton)
06_5CH	Intel Atom processors based on Goldmont microarchitecture

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_4CH	Intel Atom processor X7-Z8000 and X5-Z8000 series based on Airmont microarchitecture
06_5DH	Intel Atom processor X3-C3000 based on Silvermont microarchitecture
06_5AH	Intel Atom processor Z3500 series
06_4AH	Intel Atom processor Z3400 series
06_37H	Intel Atom processor E3000 series, Z3600 series, Z3700 series
06_4DH	Intel Atom processor C2000 series
06_36H	Intel Atom processor S1000 Series
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	Intel Atom processor family, Intel Atom processor D2000, N2000, E2000, Z2000, C1000 series
0F_06H	Intel Xeon processor 7100, 5000 Series, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
0F_03H, 0F_04H	Intel Xeon processor, Intel Xeon processor MP, Intel Pentium 4, Pentium D processors
06_09H	Intel Pentium M processor
0F_02H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
0F_0H, 0F_01H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
06_7H, 06_08H, 06_0AH, 06_0BH	Intel Pentium III Xeon processor, Intel Pentium III processor
06_03H, 06_05H	Intel Pentium II Xeon processor, Intel Pentium II processor
06_01H	Intel Pentium Pro processor
05_01H, 05_02H, 05_04H	Intel Pentium processor, Intel Pentium processor with MMX Technology
06_BEH	Intel® Processor and Intel® Core™ i3 and Intel® Core™ 3 N-Series Processors and Intel Atom® x7000 Processor Series based on Gracemont microarchitecture

The Intel® Quark™ SoC X1000 processor can be identified by the signature of DisplayFamily_DisplayModel = 05_09H and SteppingID = 0

2.1 ARCHITECTURAL MSRS

Many MSRs have carried over from one generation of IA-32 processors to the next and to Intel 64 processors. A subset of MSRs and associated bit fields, which do not change on future processor generations, are now considered architectural MSRs. For historical reasons (beginning with the Pentium 4 processor), these “architectural MSRs” were given the prefix “IA32_”. Table 2-2 lists the architectural MSRs, their addresses, their current names, their names in previous IA-32 processors, and bit fields that are considered architectural. MSR addresses outside Table 2-2 and certain bit fields in an MSR address that may overlap with architectural MSR addresses are model-specific. Code that accesses a model-specific MSR and that is executed on a processor that does not support that MSR will generate an exception.

Architectural MSR or individual bit fields in an architectural MSR may be introduced or transitioned at the granularity of certain processor family/model or the presence of certain CPUID feature flags. The right-most column of Table 2-2 provides information on the introduction of each architectural MSR or its individual fields. This information is expressed either as signature values of “DF_DM” (see Table 2-1) or via CPUID flags.

Certain bit field position may be related to the maximum physical address width, the value of which is expressed as “MAXPHYADDR” in Table 2-2. MAXPHYADDR is derived from the value enumerated in CPUID.80000008H:EAX[7:0] (this width is at most 52). However, if IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 35 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3); the value is not reduced in SEAM.¹

MSR address range between 40000000H - 4000FFFFH is marked as a specially reserved range. All existing and future processors will not implement any features using any MSR in this range.

Table 2-2. IA-32 Architectural MSRs

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: 0H, 0		IA32_P5_MC_ADDR (P5_MC_ADDR)	
See Section 2.23, "MSRs in Pentium Processors."			Pentium Processor (05_01H)
Register Address: 1H, 1		IA32_P5_MC_TYPE (P5_MC_TYPE)	
See Section 2.23, "MSRs in Pentium Processors."			DF_DM = 05_01H
Register Address: 6H, 6		IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, "Monitor/Mwait Address Range Determination."			0F_03H
Register Address: 10H, 16		IA32_TIME_STAMP_COUNTER (TSC)	
See Section 20.17, "Time-Stamp Counter."			05_01H
Register Address: 17H, 23		IA32_PLATFORM_ID (MSR_PLATFORM_ID)	
Platform ID (R/O) The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.			06_01H
49:0	Reserved.		
52:50	Platform ID (R/O) Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7		
63:53	Reserved.		
Register Address: 1BH, 27		IA32_APIC_BASE (APIC_BASE)	
This register holds the APIC base address, permitting the relocation of the APIC memory map. See Section 13.4.4, "Local APIC Status and Location," and Section 13.4.5, "Relocating the Local APIC Registers."			06_01H
7:0	Reserved.		
8	BSP Flag (R/W)		
9	Reserved.		
10	Enable x2APIC mode.		06_1AH
11	APIC Global Enable (R/W)		

1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MAXAPICADDR -1:12	APIC Base (R/W)	MAXAPICADDR is normally CPUID.80000008H:EAX[7:0] for processors that support CPUID.80000008H and 36 otherwise. If IA32_TME_ACTIVATE[39:36] > 0, MAXAPICADDR is reduced by IA32_TME_ACTIVATE[35:32].
63: MAXAPICADDR	Reserved.	
Register Address: 2FH, 47		IA32_BARRIER
IA32_BARRIER (R/O) The IA32_BARRIER MSR ensures ordered execution by acting like LFENCE, controlling the sequencing of subsequent MSR reads after prior MSR reads and instructions.		CPUID.07H.01H:EAX[27] = 1
31:0	DATA Reserved. Always 0.	
63:32	Reserved.	
Register Address: 3AH, 58		IA32_FEATURE_CONTROL
Control Features in Intel 64 Processor (R/W)		If any one enumeration condition for defined bit field holds.
0	Lock bit (R/WO): (1 = locked). When set, locks this MSR from being written; writes to this bit will result in GP(0). Note: Once the Lock bit is set, the contents of this register cannot be modified. Therefore the lock bit must be set after configuring support for Intel Virtualization Technology and prior to transferring control to an option ROM or the OS. Hence, once the Lock bit is set, the entire IA32_FEATURE_CONTROL contents are preserved across RESET when PWRGOOD is not deasserted.	If any one enumeration condition for defined bit field position greater than bit 0 holds.
1	Enable VMX inside SMX operation (R/WL) This bit enables a system executive to use VMX in conjunction with SMX to support Intel® Trusted Execution Technology. BIOS must set this bit only when CPUID.01H:ECX[6:5] returns 11b (SMX and VMX respectively).	If CPUID.01H:ECX[5] = 1 && CPUID.01H:ECX[6] = 1
2	Enable VMX outside SMX operation (R/WL) This bit enables VMX for a system executive that does not require SMX. BIOS must set this bit only when CPUID.01H:ECX.VMX[5] is set.	If CPUID.01H:ECX[5] = 1
7:3	Reserved.	
14:8	SENTER Local Function Enables (R/WL) When set, each bit in the field represents an enable control for a corresponding SENTER function. This field is supported only if CPUID.01H:ECX[6] is set.	If CPUID.01H:ECX[6] = 1
15	SENTER Global Enable (R/WL) This bit must be set to enable SENTER leaf functions. This bit is supported only if CPUID.01H:ECX[6] is set.	If CPUID.01H:ECX[6] = 1
16	Reserved.	
17	SGX Launch Control Enable (R/WL) This bit must be set to enable runtime re-configuration of SGX Launch Control via the IA32_SGXLEPUBKEYHASHn MSR.	If CPUID.07H.00H:ECX[30] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
18	SGX Global Enable (R/WL) This bit must be set to enable SGX leaf functions.	If CPUID.07H.00H:EBX[2] = 1
19	Reserved.	
20	LMCE On (R/WL) When set, system software can program the MSRs associated with LMCE to configure delivery of some machine check exceptions to a single logical processor.	If IA32_MCG_CAP[27] = 1
63:21	Reserved.	
Register Address: 3BH, 59		IA32_TSC_ADJUST
Per Logical Processor TSC Adjust (R/Write to clear)		If CPUID.07H.00H:EBX[1] = 1
63:0	THREAD_ADJUST Local offset value of the IA32_TSC for a logical processor. Reset value is zero. A write to IA32_TSC will modify the local offset in IA32_TSC_ADJUST and the content of IA32_TSC, but does not affect the internal invariant TSC hardware.	
Register Address: 48H, 72		IA32_SPEC_CTRL
Speculation Control (R/W) The MSR bits are defined as logical processor scope. On some core implementations, the bits may impact sibling logical processors on the same core. This MSR has a value of 0 after reset and is unaffected by INIT# or SIPI#.		If any one of the enumeration conditions for defined bit field positions holds.
0	Indirect Branch Restricted Speculation (IBRS). Restricts speculation of indirect branch.	If CPUID.07H.00H:EDX[26] = 1
1	Single Thread Indirect Branch Predictors (STIBP). Prevents indirect branch predictions on all logical processors on the core from being controlled by any sibling logical processor in the same core.	If CPUID.07H.00H:EDX[27] = 1
2	Speculative Store Bypass Disable (SSBD) delays speculative execution of a load until the addresses for all older stores are known.	If CPUID.07H.00H:EDX[31] = 1
3	IPRED_DIS_U If 1, enables IPRED_DIS control for CPL3.	If CPUID.07H.02H:EDX[1] = 1
4	IPRED_DIS_S If 1, enables IPRED_DIS control for CPL0/1/2.	If CPUID.07H.02H:EDX[1] = 1
5	RRSBA_DIS_U If 1, disables RRSBA behavior for CPL3.	If CPUID.07H.02H:EDX[2] = 1
6	RRSBA_DIS_S If 1, disables RRSBA behavior for CPL0/1/2.	If CPUID.07H.02H:EDX[2] = 1
7	PSFD If 1, disables Fast Store Forwarding Predictor. Note that setting bit 2 (SSBD) also disables this.	If CPUID.07H.02H:EDX[0] = 1
8	DDPD_U If 1, disables the Data Dependent Prefetcher that examines data values in memory while CPL = 3. Note that setting bit 2 (SSBD) also disables this.	If CPUID.07H.02H:EDX[3] = 1
9	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
10	BHI_DIS_S When '1, enables BHI_DIS_S behavior.	If CPUID.07H.02H:EDX[4] = 1
63:11	Reserved.	
Register Address: 49H, 73		IA32_PRED_CMD
Prediction Command (W0) Gives software a way to issue commands that affect the state of predictors.		If any one of the enumeration conditions for defined bit field positions holds.
0	Indirect Branch Prediction Barrier (IBPB)	If CPUID.07H.00H:EDX[26] = 1
63:1	Reserved.	
Register Address: 4EH, 78		IA32_PPIN_CTL
Protected Processor Inventory Number Enable Control (R/W)		If CPUID.07H.01H:EBX[0] = 1 ¹
0	LockOut (R/W0) If 0, indicates that further writes to IA32_PPIN_CTL is allowed. If 1, indicates that further writes to IA32_PPIN_CTL is disallowed. Writing 1 to this bit is only permitted if the Enable_PPIN bit is clear. The Privileged System Software Inventory Agent should read IA32_PPIN_CTL[bit 1] to determine if IA32_PPIN is accessible. The Privileged System Software Inventory Agent is not expected to write to this MSR.	
1	Enable_PPIN (R/W) If 1, indicates that IA32_PPIN is accessible using RDMSR. If 0, indicates that IA32_PPIN is inaccessible using RDMSR. Any attempt to read IA32_PPIN will cause #GP.	
63:2	Reserved.	
Register Address: 4FH, 79		IA32_PPIN
Protected Processor Inventory Number (R/O)		If CPUID.07H.01H:EBX[0] = 1 ¹
63:0	Protected Processor Inventory Number (R/O) A unique value within a given CPUID family/model/stepping signature that a privileged inventory initialization agent can access to identify each physical processor, when access to IA32_PPIN is enabled. Access to IA32_PPIN is permitted only if IA32_PPIN_CTL[bits 1:0] = '10b'.	
Register Address: 79H, 121		IA32_BIOS_UPDT_TRIG (BIOS_UPDT_TRIG)
BIOS Update Trigger (W) Executing a WRMSR instruction to this MSR causes a microcode update to be loaded into the processor. See Section 12.11.6, "Microcode Update Loader." A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.		06_01H
Register Address: 7AH, 122		IA32_FEATURE_ACTIVATION
Feature Activation (R/W) Implements Feature Activation command. WRMSR to this address activates all 'activatable' features on this thread.		
0	SE Secure Enclaves feature activation.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
1	KL Keylocker feature activation.	
63:2	Reserved.	
Register Address: 7BH, 123		IA32_MCU_ENUMERATION
IA32_MCU_ENUMERATION (R/O) Enumeration of architectural features.		
0	UNIFORM_MCU_AVAIL When set to 1, uniform microcode update is available, and UNIFORM_MCU_SCOPE (bits [10:8]) indicates the scope of writes to IA32_BIOS_UPDT_TRIG. When set to 0, uniform microcode update is not available, and writes to IA32_BIOS_UPDT_TRIG are core scoped.	
1	UNIFORM_MCU_CONFIG_REQD When set to 1, indicates that configuration is required to ensure that all MCU components are updated on WRMSR 79H, and UNIFORM_MCU_CONFIG_COMPLETE (bit 2) should be checked to determine whether the necessary configuration has been completed. When set to 0, indicates that no configuration is required, and UNIFORM_MCU_CONFIG_COMPLETE should be ignored.	
2	UNIFORM_MCU_CONFIG_COMPLETE If UNIFORM_MCU_CONFIG_REQD (bit 1) is 0, then this bit should be ignored. If UNIFORM_MCU_CONFIG_REQD is 1, then this bit indicates whether all necessary configurations have been completed to ensure that all MCU components will be updated on WRMSR 79H.	
3	ARCH_ROLLBACK_SVN_COMMIT When set to 1, indicates support for the MCU deferred SVN architecture, SVN reporting architecture, and MCU rollback architecture.	
4	MCU_STAGING When set to 1, indicates that the microcode update staging capability is supported by the processor. When supported, the use of the MCU staging capability is recommended to reduce the latency of the IA32_BIOS_UPDT_TRIG operation.	
7:5	Reserved for future use.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
15:8	UNIFORM_MCU_SCOPE Indicates the current* uniform microcode update scope: 0x02: Core Scoped 0x03: Module Scoped** 0x04: Tile Scoped** 0x05: Die Scoped** 0x80: Package Scoped 0xC0: Platform Scoped All others: Reserved for future use * The value of this field reflects the state of platform configuration and may change as the configuration changes during the boot process. Once configuration is complete, it is not expected to change during runtime. ** If these domains are enumerated by CPUID.1FH, then this field may also report them as appropriate.	
63:16	Reserved for future use.	
Register Address: 7CH, 124		IA32_MCU_STATUS
MCU Status (R/O) Communicates results from the previous patch loads.		
0	MCU_PARTIAL_UPDATE When set to 1, indicates that the most recent write to IA32_BIOS_UPDT_TRIG resulted in a partial update. This means that microcode update components were only partially updated after some portion of the MCU had already been committed and the Revision ID had been updated.	
1	AUTH_FAIL_ON_MCU_COMPONENT When set to 1, indicates that an authentication failure occurred on some portion of the MCU after another portion of the MCU had already been committed and the Revision ID had already been updated on the most recent write to IA32_BIOS_UPDT_TRIG.	
2	Reserved for future use.	
3	POST_BIOS_MCU When set to 1, indicates that an update was successfully loaded via IA32_BIOS_UPDT_TRIG after bit 0 of MSR_BIOS_DONE (address 151H) was set to 1.	
63:4	Reserved for future use.	
Register Address: 82H, 130		IA32_FZM_RANGE_INDEX
IA32_FZM_RANGE_INDEX (R/W) Index and Domain handle for a valid FZM region. Programmed by software and used by other FRM MSRs FZM Range Index register to R/W Domain Index.		
3:0	REGION_INDEX Holds the Index of domain.	
7:4	Reserved.	
12:8	DOMAIN_HANDLE Holds the Domain Handle.	
63:13	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 83H, 131		IA32_FZM_DOMAIN_CONFIG
IA32_FZM_DOMAIN_CONFIG (R/O) Bit mask of valid regions within the domain identified by FZM_RANGE_INDEX.		
63:0	REGION_BITMAP Bitmap of valid regions for a given domain.	
Register Address: 84H, 132		IA32_FZM_RANGE_STARTADDR
IA32_FZM_RANGE_STARTADDR (R/O) Start address of the FZM range pointed to by FZM_RANGE_INDEX.		
51:0	START_ADDR Start address of the specified domain in FZM_RANGE_INDEX.	
63:52	Reserved.	
Register Address: 85H, 133		IA32_FZM_RANGE_ENDADDR
IA32_FZM_RANGE_ENDADDR (R/O) End address of the specified domain in FZM_RANGE_INDEX.		
51:0	END_ADDR End address of the specified domain in FZM_RANGE_INDEX.	
63:52	Reserved.	
Register Address: 86H, 134		IA32_FZM_RANGE_WRITESTATUS
IA32_FZM_RANGE_WRITESTATUS (R/O) Write status of the FZM range pointed to by FZM_RANGE_INDEX.		
0	WRITE_STATUS Write status of the specified domain in FZM_RANGE_INDEX.	
1	READ_STATUS Read status of the specified domain in FZM_RANGE_INDEX.	
63:2	Reserved.	
Register Address: 87H, 135		IA32_MKTME_KEYID_PARTITIONING
MKTME KEY ID Partitioning (R/O) Enumerates the number of activated KeyIDs for Intel TME-MK and Intel TDX.		
31:0	NUM_MKTME_KEYIDS Number of activated Intel TME-MK KeyIDs. This field is supported on all parts that enumerate support for Intel Total Memory Encryption - Multi-Key (Intel TME-MK). If IA32_TME_ACTIVATE.LOCK is 1, this field reports the number of shared KeyIDs supported by the processor (all KeyIDs other than SEAM-private KeyIDs and KeyID 0); otherwise, it reports 0. Intel TME-MK KeyIDs will always span the KeyID range [1 ... NUM_MKTME_KEYIDS]. Note: KeyID 0 is reserved for TME and will not be included.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:32	NUM_TDX_PRIV_KEYIDS Number of activated SEAM-private KeyIDs. This field is supported on all parts that enumerate support for SEAM mode. If IA32_TME_ACTIVATE.LOCK is 1, this field reports the number of SEAM-private KeyIDs supported by the processor (all KeyIDs, except shared KeyIDs and KeyID 0); otherwise, it reports 0. SEAM-private KeyIDs will always span the range [NUM_MKTME_KEYIDS+1... (NUM_MKTME_KEYIDS + NUM_TDX_PRIV_KEYIDS)]. Note: KeyID 0 is reserved for TME and will not be included.	
Register Address: 8BH, 139		IA32_BIOS_SIGN_ID (BIOS_SIGN/BBL_CR_D3)
BIOS Update Signature (R/W) Returns the microcode update signature following the execution of CPUID.01H. A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.		06_01H
31:0	Reserved.	
63:32	PATCH_SIGN_ID It is recommended that this field be preloaded with zero prior to executing CPUID. If the field remains zero following the execution of CPUID, this indicates that no microcode update is loaded. Any non-zero value is the microcode update signature patch signature ID.	
Register Address: 8CH, 140		IA32_SGXLEPUBKEYHASH0
IA32_SGXLEPUBKEYHASH[63:0] (R/W) Bits 63:0 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Read permitted if CPUID.12H.00H:EAX[0] = 1 && CPUID.07H.00H:ECX[30] = 1. Write permitted if CPUID.12H.00H:EAX[0] = 1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1.
Register Address: 8DH, 141		IA32_SGXLEPUBKEYHASH1
IA32_SGXLEPUBKEYHASH[127:64] (R/W) Bits 127:64 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 8EH, 142		IA32_SGXLEPUBKEYHASH2
IA32_SGXLEPUBKEYHASH[191:128] (R/W) Bits 191:128 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 8FH, 143		IA32_SGXLEPUBKEYHASH3
IA32_SGXLEPUBKEYHASH[255:192] (R/W) Bits 255:192 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 90H, 144		IA32_SGXLEPUBKEYHASH4
IA32_SGXLEPUBKEYHASH[319:256] (R/W) Bits 319:256 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 91H, 145		IA32_SGXLEPUBKEYHASH5
IA32_SGXLEPUBKEYHASH[383:320] (R/W) Bits 383:320 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 9BH, 155		IA32_SMM_MONITOR_CTL
SMM Monitor Configuration (R/W)		If CPUID.01H:ECX[5] = 1 CPUID.01H:ECX[6] = 1
0	Valid (R/W)	
1	Reserved.	
2	Controls SMI unblocking by VMXOFF (see Section 34.14.4).	If IA32_VMX_MISC[28]
11:3	Reserved.	
31:12	MSEG Base (R/W)	
63:32	Reserved.	
Register Address: 9EH, 158		IA32_SMBASE
Base address of the logical processor's SMRAM image (R/O, SMM only).		If IA32_VMX_MISC[15]
Register Address: BCH, 188		IA32_MISC_PACKAGE_CTL5
Power Filtering Control (R/W) This MSR has a value of 0 after reset and is unaffected by INIT# or SIPI#.		If IA32_ARCH_CAPABILITIES [10] = 1
0	ENERGY_FILTERING_ENABLE (R/W) If set, RAPL MSRs report filtered processor power consumption data. This bit can be changed from 0 to 1, but cannot be changed from 1 to 0. After setting, all attempts to clear it are ignored until the next processor reset.	If IA32_ARCH_CAPABILITIES [11] = 1
63:1	Reserved.	
Register Address: BDH, 189		IA32_XAPIC_DISABLE_STATUS
xAPIC Disable Status (R/O)		If CPUID.07H:00H:EDX[29] = 1 and IA32_ARCH_CAPABILITIES [21] = 1
0	LEGACY_XAPIC_DISABLED When set, indicates that the local APIC is in x2APIC mode (IA32_APIC_BASE.EXTD = 1) and that attempts to clear IA32_APIC_BASE.EXTD will fail (e.g., WRMSR will #GP).	
63:1	Reserved.	
Register Address: C1H, 193		IA32_PMC0 (PERFCTR0)
General Performance Counter 0 (R/W)		If CPUID.0AH:EAX[15:8] > 0
Register Address: C2H, 194		IA32_PMC1 (PERFCTR1)
General Performance Counter 1 (R/W)		If CPUID.0AH:EAX[15:8] > 1
Register Address: C3H, 195		IA32_PMC2
General Performance Counter 2 (R/W)		If CPUID.0AH:EAX[15:8] > 2
Register Address: C4H, 196		IA32_PMC3
General Performance Counter 3 (R/W)		If CPUID.0AH:EAX[15:8] > 3

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: C5H, 197		IA32_PMC4
General Performance Counter 4 (R/W)		If CPUID.0AH:EAX[15:8] > 4
Register Address: C6H, 198		IA32_PMC5
General Performance Counter 5 (R/W)		If CPUID.0AH:EAX[15:8] > 5
Register Address: C7H, 199		IA32_PMC6
General Performance Counter 6 (R/W)		If CPUID.0AH:EAX[15:8] > 6
Register Address: C8H, 200		IA32_PMC7
General Performance Counter 7 (R/W)		If CPUID.0AH:EAX[15:8] > 7
Register Address: C9H, 201		IA32_PMC8
General Performance Counter 8 (R/W)		If CPUID.0AH:EAX[15:8] > 8
Register Address: CAH, 202		IA32_PMC9
General Performance Counter 9 (R/W)		If CPUID.0AH:EAX[15:8] > 9
Register Address: CFH, 207		IA32_CORE_CAPABILITIES
IA32 Core Capabilities Register		If CPUID.07H.00H:EDX[30] = 1
63:0	Reserved.	No architecturally defined bits.
Register Address: E1H, 225		IA32_UMWAIT_CONTROL
UMWAIT Control (R/W)		
0	C0.2 is not allowed by the OS. Value of "1" means all C0.2 requests revert to C0.1.	
1	Reserved.	
31:2	Determines the maximum time in TSC-quanta that the processor can reside in either C0.1 or C0.2. A zero value indicates no maximum time. The maximum time value is a 32-bit value where the upper 30 bits come from this field and the lower two bits are zero.	
Register Address: E7H, 231		IA32_MPERF
TSC Frequency Clock Counter (R/Write to clear)		If CPUID.06H:ECX[0] = 1
63:0	CO_MCNT: C0 TSC Frequency Clock Count Increments at fixed interval (relative to TSC freq.) when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_APERF.	
Register Address: E8H, 232		IA32_APERF
Actual Performance Clock Counter (R/Write to clear)		If CPUID.06H:ECX[0] = 1
63:0	CO_ACNT: C0 Actual Frequency Clock Count Accumulates core clock counts at the coordinated clock frequency, when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_MPERF.	
Register Address: FEH, 254		IA32_MTRRCAP (MTRRcap)
MTRR Capability (R/O) See Section 14.11.2.1, "IA32_MTRR_DEF_TYPE MSR."		06_01H
7:0	VCNT: The number of variable memory type ranges in the processor.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
8	Fixed range MTRRs are supported when set.	
9	Reserved.	
10	WC supported when set.	
11	SMRR supported when set.	
12	PRMRR supported when set.	
14:13	Reserved.	
15	SEAMRR supported when set.	
63:16	Reserved.	
Register Address: 10AH, 266		IA32_ARCH_CAPABILITIES
Enumeration of Architectural Features (R/O)		If CPUID.07H.00H:EDX[29] = 1
0	RDCL_NO: The processor is not susceptible to Rogue Data Cache Load (RDCL).	
1	IBRS_ALL: The processor supports enhanced IBRS.	
2	RSBA: The processor supports RSB Alternate. Alternative branch predictors may be used by RET instructions when the RSB is empty. SW using retpoline may be affected by this behavior.	
3	SKIP_L1DFL_VMENTRY: A value of 1 indicates the hypervisor need not flush the L1D on VM entry.	
4	SSB_NO: Processor is not susceptible to Speculative Store Bypass.	
5	MDS_NO: Processor is not susceptible to Microarchitectural Data Sampling (MDS).	
6	IF_PSCCHANGE_MC_NO: The processor is not susceptible to a machine check error due to modifying the size of a code page without TLB invalidation.	
7	TSX_CTRL: If 1, indicates presence of IA32_TSX_CTRL MSR.	
8	TAA_NO: If 1, processor is not affected by TAA.	
9	MCU_CONTROL: If 1, the processor supports the IA32_MCU_CONTROL MSR.	
10	MISC_PACKAGE_CTL: The processor supports IA32_MISC_PACKAGE_CTL MSR.	
11	ENERGY_FILTERING_CTL: The processor supports setting and reading the IA32_MISC_PACKAGE_CTL[0] (ENERGY_FILTERING_ENABLE) bit.	
12	DOITM: If 1, the processor supports Data Operand Independent Timing Mode.	
13	SBDR_SSDP_NO: The processor is not affected by either the Shared Buffers Data Read (SBDR) vulnerability or the Sideband Stale Data Propagator (SSDP).	
14	FBSDP_NO: The processor is not affected by the Fill Buffer Stale Data Propagator (FBSDP).	
15	PSDP_NO: The processor is not affected by vulnerabilities involving the Primary Stale Data Propagator (PSDP).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
16	MCU_ENUMERATION: If 1, the processor supports the IA32_MCU_ENUMERATION and IA32_MCU_STATUS MSRs.	
17	FB_CLEAR: If 1, the processor supports overwrite of fill buffer values as part of MD_CLEAR operations with the VERW instruction. On these processors, L1D_FLUSH does not overwrite fill buffer values.	
18	FB_CLEAR_CTRL: If 1, the processor supports the IA32_MCU_OPT_CTRL MSR and allows software to set bit 3 of that MSR (FB_CLEAR_DIS).	
19	RRSBA: A value of 1 indicates the processor may have the RRSBA alternate prediction behavior, if not disabled by RRSBA_DIS_U or RRSBA_DIS_S.	
20	BHI_NO: A value of 1 indicates BHI_NO branch prediction behavior, regardless of the value of IA32_SPEC_CTRL[BHI_DIS_S] MSR bit.	
21	XAPIC_DISABLE_STATUS: Enumerates that the IA32_XAPIC_DISABLE_STATUS MSR exists, and that bit 0 specifies whether the legacy xAPIC is disabled and APIC state is locked to x2APIC.	
22	MCU_EXTENDED_SERVICE: If 1, the processor supports MCU Extended servicing - IA32_MCU_EXT_SERVICE MSR.	
23	OVERCLOCKING_STATUS: If set, the IA32_OVERCLOCKING_STATUS MSR exists.	
24	PBSRB_NO: If 1, the processor is not affected by issues related to Post-Barrier Return Stack Buffer Predictions.	
25	GDS_CTRL: If 1, the processor supports the GDS_MITG_DIS and GDS_MITG_LOCK bits of the IA32_MCU_OPT_CTRL MSR.	
26	GDS_NO: If 1, the processor is not affected by Gather Data Sampling.	
27	RFDS_NO: If 1, the processor is not affected by Register File Data Sampling.	
28	RFDS_CLEAR: If 1, when VERW is executed the processor will clear stale data from register files affected by Register File Data Sampling.	
29	IGN_UMONITOR_SUPPORT If 0, IA32_MCU_OPT_CTRL bit 6 (IGN_UMONITOR) is not supported. If 1, it indicates support of IA32_MCU_OPT_CTRL bit 6 (IGN_UMONITOR).	
30	MON_UMON_MITG_SUPPORT If 0, IA32_MCU_OPT_CTRL bit 7 (MON_UMON_MITG) is not supported. If 1, it indicates support of IA32_MCU_OPT_CTRL bit 7 (MON_UMON_MITG).	
31	Reserved.	
32	PBOPT_SUPPORT If 0, IA32_PBOPT_CTRL bit 0 (Prediction Barrier Option [PBOPT]) is not supported. If 1, IA32_PBOPT_CTRL bit 0 (Prediction Barrier Option [PBOPT]) is supported.	
61:33	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
62	ITS_NO If 0, the hypervisor indicates that the system is not affected by Indirect Target Selection. If 1, the hypervisor indicates that the system may be affected by Indirect Target Selection.		
63	Reserved.		
Register Address: 10BH, 267		IA32_FLUSH_CMD	
Flush Command (W0) Gives software a way to invalidate structures with finer granularity than other architectural methods.		If any one of the enumeration conditions for defined bit field positions holds.	
0	L1D_FLUSH Writeback and invalidate the L1 data cache.		If CPUID.07H.00H:EDX[28] = 1
63:1	Reserved.		
Register Address: 10FH, 271		IA32_TSX_FORCE_ABORT	
TSX Force Abort		If CPUID.07H.00H:EDX[13] = 1	
0	RTM_FORCE_ABORT If 1, all RTM transactions abort with EAX code 0.		R/W, Default: 0 If CPUID.07H.00H:EDX[11] = 1, bit 0 is always 1 and writes to change it are ignored. If SDV_ENABLE_RTM is 1, bit 0 is always 0 and writes to change it are ignored.
1	TSX_CPUID_CLEAR When set, CPUID.07H.00H:EBX[11] = 0 and CPUID.07H.00H:EBX[4] = 0.		R/W, Default: 0 Can be set only if CPUID.07H.00H:EDX[11] = 1 or if SDV_ENABLE_RTM is 1.
2	SDV_ENABLE_RTM When set, CPUID.07H.00H:EDX[11] = 0 and the processor may not force abort RTM. This unsupported mode should only be used for software development and not for production usage.		R/W, Default: 0 If 0, can be set only if CPUID.07H.00H:EDX[11] = 1.
63:3	Reserved.		
Register Address: 122H, 290		IA32_TSX_CTRL	
IA32_TSX_CTRL (R/W)		Thread scope. Not architecturally serializing. Available when CPUID.07H.00H:EDX.ARCH_CAPABILITIES[29] = 1 and IA32_ARCH_CAPABILITIES.bit 7 = 1.	
0	RTM_DISABLE When set to 1, XBEGIN will always abort with EAX code 0.		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
1	TSX_CPUID_CLEAR When set to 1, CPUID.07H.00H:EBX.RTM[11] and CPUID.07H.00H:EBX.HLE[4] report 0. When set to 0 and the SKU supports TSX, these bits will return 1.	
63:2	Reserved.	
Register Address: 123H, 291		IA32_MCU_OPT_CTRL
Microcode Update Option Control (R/W)		If CPUID.07H.00H:EDX[9] = 1 or CPUID.07H.00H:EDX[11] = 1 IA32_ARCH_CAPABILITIES [18] = 1 or IA32_ARCH_CAPABILITIES [25] = 1 or IA32_ARCH_CAPABILITIES [29] = 1 or IA32_ARCH_CAPABILITIES [30] = 1
0	RNGDS_MITG_DIS (R/W) If 0 (default), SRBDS mitigation is enabled for RDRAND and RDSEED. If 1, SRBDS mitigation is disabled for RDRAND and RDSEED executed outside of Intel SGX enclaves.	If CPUID.07H.00H:EDX[9] = 1
1	RTM_ALLOW If 0, XBEGIN will always abort with EAX code 0. If 1, XBEGIN behavior depends on the value of IA32_TSX_CTRL[RTM_DISABLE].	If CPUID.07H.00H:EDX[11] = 1 Read/Write Setting RTM_LOCKED prevents writes to this bit.
2	RTM_LOCKED When 1, RTM_ALLOW is locked at zero, writes to RTM_ALLOW will be ignored.	If CPUID.07H.00H:EDX[11] = 1 Read-Only status bit.
3	FB_CLEAR_DIS If 1, prevents the VERW instruction from performing an FB_CLEAR action.	If IA32_ARCH_CAPABILITIES [18] = 1
4	GDS_MITG_DIS If 0, the Gather Data Sampling mitigation is enabled (patch load time default). If 1 on all threads for a given core, the Gather Data Sampling mitigation is disabled.	If IA32_ARCH_CAPABILITIES [25] = 1
5	GDS_MITG_LOCK If 0, not locked, and GDS_MITG_DIS is under OS control. If 1, locked and GDS_MITG_DIS is forced to 0 (writes are ignored).	If IA32_ARCH_CAPABILITIES [25] = 1
6	IGN_UMONITOR If 0, enable CPL0-3 software to use the UMONITOR/UMWAIT instructions. If 1 (default), disable UMONITOR functionality. CPL0-3 software will be able to call the UMONITOR instruction without causing a fault, however the address monitoring hardware will not be armed. When UMWAIT is called, it will not enter an implementation-dependent optimized state.	If IA32_ARCH_CAPABILITIES [29] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
7	MON_UMON_MITG If 0 (default), disabled. If 1, enable: Flush the thread's previously monitored address from the CPU caches as part of the (U)MONITOR instruction. Additionally, for every 4th (U)MONITOR instruction within a core, flush the peer hyperthread's monitored address from the CPU caches as well. This will increase the latency of the instruction. This may have a minor impact on workloads using the (U)MONITOR instruction.	If IA32_ARCH_CAPABILITIES [30] = 1
63:8	Reserved.	
Register Address: 174H, 372		IA32_SYSENTER_CS
SYSENTER_CS_MSR (R/W)		06_01H
15:0	CS Selector.	
31:16	Not used.	Can be read and written.
63:32	Not used.	Writes ignored; reads return zero.
Register Address: 175H, 373		IA32_SYSENTER_ESP
SYSENTER_ESP_MSR (R/W)		06_01H
Register Address: 176H, 374		IA32_SYSENTER_EIP
SYSENTER_EIP_MSR (R/W)		06_01H
Register Address: 179H, 377		IA32_MCG_CAP (MCG_CAP)
Global Machine Check Capability (R/O)		06_01H
7:0	Count: Number of reporting banks.	
8	MCG_CTL_P: IA32_MCG_CTL is present if this bit is set.	
9	MCG_EXT_P: Extended machine check state registers are present if this bit is set.	
10	MCP_CMCI_P: Support for corrected MC error event is present.	06_01H
11	MCG_TES_P: Threshold-based error status register are present if this bit is set.	
12	MCG_SEAM_NP: Indicates that the processor supports SEAM non-root operation indication in IA32_MCG_STATUS.	
15:13	Reserved.	
23:16	MCG_EXT_CNT: Number of extended machine check state registers present.	
24	MCG_SER_P: The processor supports software error recovery if this bit is set.	
25	Reserved.	
26	MCG_ELOG_P: Indicates that the processor allows platform firmware to be invoked when an error is detected so that it may provide additional platform specific information in an ACPI format "Generic Error Data Entry" that augments the data included in machine check bank registers.	06_3EH
27	MCG_LMCE_P: Indicates that the processor supports extended state in IA32_MCG_STATUS and associated MSR necessary to configure Local Machine Check Exception (LMCE).	06_3EH

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:28	Reserved.	
Register Address: 17AH, 378		IA32_MCG_STATUS (MCG_STATUS)
Global Machine Check Status (R/W)		06_01H
0	RIPV. Restart IP valid.	06_01H
1	EIPV. Error IP valid.	06_01H
2	MCIP. Machine check in progress.	06_01H
3	LMCE_S. Local machine check.	If IA32_MCG_CAP.LMCE_P[27] = 1
11:4	Reserved.	
12	SEAM_NR. SEAM non-root operation.	If IA32_MCG_CAP.SEAM_NR_P[12] = 1
63:13	Reserved.	
Register Address: 17BH, 379		IA32_MCG_CTL (MCG_CTL)
Global Machine Check Control (R/W)		If IA32_MCG_CAP.CTL_P[8] = 1
Register Address: 180H–185H, 384–389		N/A
Reserved		06_0EH ²
Register Address: 186H, 390		IA32_PERFVTSEL0 (PERFVTSEL0)
Performance Event Select Register 0 (R/W)		If CPUID.0AH:EAX[15:8] > 0
7:0	Event Select: Selects a performance event logic unit.	
15:8	UMask: Qualifies the microarchitectural condition to detect on the selected event logic.	
16	USR: Counts while in privilege level is not ring 0.	
17	OS: Counts while in privilege level is ring 0.	
18	Edge: Enables edge detection if set.	
19	PC: Enables pin control.	
20	INT: Enables interrupt on counter overflow.	
21	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	
22	EN: Enables the corresponding performance counter to commence counting when this bit is set.	
23	INV: Invert the CMASK.	
31:24	CMASK: When CMASK is not zero, the corresponding performance counter increments each cycle if the event count is greater than or equal to the CMASK.	
63:32	Reserved.	
Register Address: 187H, 391		IA32_PERFVTSEL1 (PERFVTSEL1)
Performance Event Select Register 1 (R/W)		If CPUID.0AH:EAX[15:8] > 1
Register Address: 188H, 392		IA32_PERFVTSEL2
Performance Event Select Register 2 (R/W)		If CPUID.0AH:EAX[15:8] > 2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 189H, 393		IA32_PERFEVTSEL3
Performance Event Select Register 3 (R/W)		If CPUID.0AH:EAX[15:8] > 3
Register Address: 18AH, 394		IA32_PERFEVTSEL4
Performance Event Select Register 4 (R/W)		If CPUID.0AH:EAX[15:8] > 4
Register Address: 18BH, 395		IA32_PERFEVTSEL5
Performance Event Select Register 5 (R/W)		If CPUID.0AH:EAX[15:8] > 5
Register Address: 18CH, 396		IA32_PERFEVTSEL6
Performance Event Select Register 6 (R/W)		If CPUID.0AH:EAX[15:8] > 6
Register Address: 18DH, 397		IA32_PERFEVTSEL7
Performance Event Select Register 7 (R/W)		If CPUID.0AH:EAX[15:8] > 7
Register Address: 18EH, 398		IA32_PERFEVTSEL8
Performance Event Select Register 8 (R/W)		If CPUID.0AH:EAX[15:8] > 8
Register Address: 18FH, 399		IA32_PERFEVTSEL9
Performance Event Select Register 9 (R/W)		If CPUID.0AH:EAX[15:8] > 9
Register Address: 18AH–194H, 394–404		N/A
Reserved.		06_0EH ³
Register Address: 195H, 405		IA32_OVERCLOCKING_STATUS
Overclocking Status (R/O) IA32_ARCH_CAPABILITIES[bit 23] enumerates support for this MSR.		
0	Overclocking Utilized Indicates if specific forms of overclocking have been enabled on this boot or reset cycle: 0 indicates no, 1 indicates yes.	
1	Undervolt Protection Indicates if the “Dynamic OC Undervolt Protection” security feature is active: 0 indicates disabled, 1 indicates enabled.	
2	Overclocking Secure Status Indicates that overclocking capabilities have been unlocked by BIOS, with or without overclocking: 0 indicates Not Secured, 1 indicates Secure.	
63:3	Reserved.	
Register Address: 196H–197H, 406–407		N/A
Reserved.		06_0EH ³
Register Address: 198H, 408		IA32_PERF_STATUS
Current Performance Status (R/O) See Section 17.1.1, “Software Interface For Initiating Performance State Transitions.”		0F_03H
15:0	Current Performance State Value.	
63:16	Reserved.	
Register Address: 199H, 409		IA32_PERF_CTL

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Performance Control MSR (R/W) Software makes a request for a new Performance state (P-State) by writing this MSR. See Section 17.1.1, "Software Interface For Initiating Performance State Transitions."		0F_03H
15:0	Target performance State Value.	
31:16	Reserved.	
32	Intel® Dynamic Acceleration Technology Engage (R/W) When set to 1: Disengages Intel Dynamic Acceleration Technology.	06_0FH (Mobile only)
63:33	Reserved.	
Register Address: 19AH, 410		IA32_CLOCK_MODULATION
Clock Modulation Control (R/W) See Section 17.8.3, "Software Controlled Clock Modulation."		If CPUID.01H:EDX[22] = 1
0	Extended On-Demand Clock Modulation Duty Cycle.	If CPUID.06H:EAX[5] = 1
3:1	On-Demand Clock Modulation Duty Cycle: Specific encoded values for target duty cycle modulation.	If CPUID.01H:EDX[22] = 1
4	On-Demand Clock Modulation Enable: Set 1 to enable modulation.	If CPUID.01H:EDX[22] = 1
63:5	Reserved.	
Register Address: 19BH, 411		IA32_THERM_INTERRUPT
Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the processor's thermal sensors and thermal monitor. See Section 17.8.2, "Thermal Monitor."		If CPUID.01H:EDX[22] = 1
0	High-Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
1	Low-Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
2	PROCHOT# Interrupt Enable	If CPUID.01H:EDX[22] = 1
3	FORCEPR# Interrupt Enable	If CPUID.01H:EDX[22] = 1
4	Critical Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
7:5	Reserved.	
14:8	Threshold #1 Value	If CPUID.01H:EDX[22] = 1
15	Threshold #1 Interrupt Enable	If CPUID.01H:EDX[22] = 1
22:16	Threshold #2 Value	If CPUID.01H:EDX[22] = 1
23	Threshold #2 Interrupt Enable	If CPUID.01H:EDX[22] = 1
24	Power Limit Notification Enable	If CPUID.06H:EAX[4] = 1
63:25	Reserved.	
Register Address: 19CH, 412		IA32_THERM_STATUS
Thermal Status Information (R/O) Contains status information about the processor's thermal sensor and automatic thermal monitoring facilities. See Section 17.8.2, "Thermal Monitor."		If CPUID.01H:EDX[22] = 1
0	Thermal Status (R/O)	If CPUID.01H:EDX[22] = 1
1	Thermal Status Log (R/W)	If CPUID.01H:EDX[22] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
2	PROCHOT # or FORCEPR# event (R/O)	If CPUID.01H:EDX[22] = 1
3	PROCHOT # or FORCEPR# log (R/WCO)	If CPUID.01H:EDX[22] = 1
4	Critical Temperature Status (R/O)	If CPUID.01H:EDX[22] = 1
5	Critical Temperature Status log (R/WCO)	If CPUID.01H:EDX[22] = 1
6	Thermal Threshold #1 Status (R/O)	If CPUID.01H:ECX[8] = 1
7	Thermal Threshold #1 log (R/WCO)	If CPUID.01H:ECX[8] = 1
8	Thermal Threshold #2 Status (R/O)	If CPUID.01H:ECX[8] = 1
9	Thermal Threshold #2 log (R/WCO)	If CPUID.01H:ECX[8] = 1
10	Power Limitation Status (R/O)	If CPUID.06H:EAX[4] = 1
11	Power Limitation log (R/WCO)	If CPUID.06H:EAX[4] = 1
12	Current Limit Status (R/O)	If CPUID.06H:EAX[7] = 1
13	Current Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
14	Cross Domain Limit Status (R/O)	If CPUID.06H:EAX[7] = 1
15	Cross Domain Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
22:16	Digital Readout (R/O)	If CPUID.06H:EAX[0] = 1
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O)	If CPUID.06H:EAX[0] = 1
31	Reading Valid (R/O)	If CPUID.06H:EAX[0] = 1
63:32	Reserved.	
Register Address: 1A0H, 416		IA32_MISC_ENABLE
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable When set, the fast-strings feature (for REP MOVSB and REP STOSB) is enabled (default). When clear, fast-strings are disabled.	OF_OH
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows the processor to automatically reduce power consumption in response to TCC activation. 0 = Disabled. Note: In some products clearing this bit might be ignored in critical thermal conditions, and TM1, TM2, and adaptive thermal throttling will still be activated. The default value of this field varies with product. See respective tables where default value is listed.	OF_OH
6:4	Reserved.	
7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.	OF_OH

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) 1 = Processor doesn't support branch trace storage (BTS). 0 = BTS is supported.	0F_0H
12	Processor Event Based Sampling (PEBS) Unavailable (R/O) 1 = PEBS is not supported. 0 = PEBS is supported.	06_0FH
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) 0 = Enhanced Intel SpeedStep Technology disabled. 1 = Enhanced Intel SpeedStep Technology enabled.	If CPUID.01H:ECX[7] = 1
17	Reserved.	
18	ENABLE MONITOR FSM (R/W) When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[3] = 0). This indicates that MONITOR/MWAIT are not supported. Software attempts to execute MONITOR/MWAIT will cause #UD when this bit is 0. When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[3] = 1). If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.	0F_03H
21:19	Reserved.	
22	Limit CPUID Maxval (R/W) When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 2.	CPUID.00H:EAX > 2 and CPUID.07H:01H:EBX. CPUIDMAXVAL_LIM_RMV[3] = 0
23	xTPR Message Disable (R/W) When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority.	If CPUID.01H:ECX[14] = 1
63:24	Reserved. Note: Some older processors defined one of these bits as a disable for the execute-disable feature of paging. If a processor supports this bit, this information is provided in the model-specific tables. See Table 2-3 for the definition of this bit.	
Register Address: 1B0H, 432		IA32_ENERGY_PERF_BIAS
Performance Energy Bias Hint (R/W)		If CPUID.06H:ECX[3] = 1
3:0	Power Policy Preference: 0 indicates preference to highest performance. 15 indicates preference to maximize energy saving.	
63:4	Reserved.	
Register Address: 1B1H, 433		IA32_PACKAGE_THERM_STATUS

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Package Thermal Status Information (R/O) Contains status information about the package's thermal sensor. See Section 17.9, "Package Level Thermal Management."		If CPUID.06H:EAX[6] = 1
0	Pkg Thermal Status (R/O)	
1	Pkg Thermal Status Log (R/W)	
2	Pkg PROCHOT # event. (R/O)	
3	Pkg PROCHOT # log. (R/WC0)	
4	Pkg Critical Temperature Status. (R/O)	
5	Pkg Critical Temperature Status Log. (R/WC0)	
6	Pkg Thermal Threshold #1 Status. (R/O)	
7	Pkg Thermal Threshold #1 Log. (R/WC0)	
8	Pkg Thermal Threshold #2 Status. (R/O)	
9	Pkg Thermal Threshold #1 Log. (R/WC0)	
10	Pkg Power Limitation Status. (R/O)	
11	Pkg Power Limitation Log. (R/WC0)	
15:12	Reserved.	
22:16	Pkg Digital Readout. (R/O)	
25:23	Reserved.	
26	Hardware Feedback Interface Structure Change Status.	If CPUID.06H:EAX[19] = 1
63:27	Reserved.	
Register Address: 1B2H, 434		IA32_PACKAGE_THERM_INTERRUPT
Pkg Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the package's thermal sensor. See Section 17.9, "Package Level Thermal Management."		If CPUID.06H:EAX[6] = 1
0	Pkg High-Temperature Interrupt Enable.	
1	Pkg Low-Temperature Interrupt Enable.	
2	Pkg PROCHOT# Interrupt Enable.	
3	Reserved.	
4	Pkg Overheat Interrupt Enable.	
7:5	Reserved.	
14:8	Pkg Threshold #1 Value.	
15	Pkg Threshold #1 Interrupt Enable.	
22:16	Pkg Threshold #2 Value.	
23	Pkg Threshold #2 Interrupt Enable.	
24	Pkg Power Limit Notification Enable.	
25	Hardware Feedback Interrupt Enable.	If CPUID.06H:EAX[19] = 1
63:26	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 1C4H, 452		IA32_XFD
Extended Feature Disable Control (R/W) Controls which XSAVE-enabled features are temporarily disabled. See Section 13.14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.		If CPUID.0DH.01H:EAX[4] = 1
Register Address: 1C5H, 453		IA32_XFD_ERR
Extended Feature Disable Error Code (R/W) Reports which XSAVE-enabled features caused a fault due to being disabled. See Section 13.14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.		If CPUID.0DH.01H:EAX[4] = 1
Register Address: 1CCH, 460		IA32_FRED_RSP0
FRED Regular Stack Pointer 0 If FRED event delivery causes a transition from CPL 3, RSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1
Register Address: 1CDH, 461		IA32_FRED_RSP1
FRED Regular Stack Pointer 1 If FRED event delivery causes a transition to stack level 1 (from CPL 0), RSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1
Register Address: 1CEH, 462		IA32_FRED_RSP2
FRED Regular Stack Pointer 2 If FRED event delivery causes a transition to stack level 2 (from CPL 0), RSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1
Register Address: 1CFH, 463		IA32_FRED_RSP3
FRED Regular Stack Pointer 3 If FRED event delivery causes a transition to stack level 3 (from CPL 0), RSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1
Register Address: 1D0H, 464		IA32_FRED_STKLVLS
FRED Stack Levels This MSR is interpreted as an array of 32 2-bit values, one for each vector in the range 0-31. For an exception with vector v (or for a non-maskable interrupt, which always has vector v = 2) that occurs in ring 0, FRED event delivery ensures that the new stack level is at least the value of IA32_FRED_STKLVLS[2v+1:2v].		If CPUID.07H.01H:EAX.FRED[17] = 1
Register Address: 1D1H, 465		IA32_FRED_SSP1
FRED Shadow-Stack Pointer 1 If shadow stacks are enabled and FRED event delivery causes a transition to stack level 1, SSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1 AND CPUID.07H.00H:ECX.CET_SS[7] = 1
2:0	Reserved.	
63:3	Address to the new stack level.	
Register Address: 1D2H, 466		IA32_FRED_SSP2
FRED Shadow-Stack Pointer 2 If shadow stacks are enabled and FRED event delivery causes a transition to stack level 2, SSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1 AND CPUID.07H.00H:ECX.CET_SS[7] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
2:0	Reserved.	
63:3	Address to the new stack level.	
Register Address: 1D3H, 467		IA32_FRED_SSP3
FRED Shadow-Stack Pointer 3 If shadow stacks are enabled and FRED event delivery causes a transition to stack level 3, SSP is loaded from this MSR.		If CPUID.07H.01H:EAX.FRED[17] = 1 AND CPUID.07H.00H:ECX.CET_SS[7] = 1
2:0	Reserved.	
63:3	Address to the new stack level.	
Register Address: 1D4H, 468		IA32_FRED_CONFIG
FRED Configuration Register		If CPUID.07H.01H:EAX.FRED[17] = 1
1:0	CSL:	
2	Reserved.	
3	SSP: If FRED event delivery is not changing stacks, setting this bit to 1 decrements the shadow-stack pointer (SSP) by 8.	
5:4	Reserved.	
8:6	The amount, measured in 64-byte cache lines, that FRED event delivery decrements the regular stack pointer when not changing stacks.	
10:9	The stack level used for maskable interrupts that are delivered in ring 0.	
11	Reserved.	
63:12	The upper bits of the linear address of a page in memory containing event handlers. FRED event delivery loads RIP to refer to an entry point on this page.	
Register Address: 1D9H, 473		IA32_DEBUGCTL (MSR_DEBUGCTLA, MSR_DEBUGCTLB)
Trace/Profile Resource Control (R/W)		06_0EH
0	LBR: Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.	06_01H
1	BTF: Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.	06_01H
2	BLD: Enable OS bus-lock detection. See Section 20.3.1.6 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.	If (CPUID.07H.00H:ECX[24] = 1)
5:3	Reserved.	
6	TR: Setting this bit to 1 enables branch trace messages to be sent.	06_0EH
7	BTS: Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.	06_0EH
8	BTINT: When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.	06_0EH
9	1: BTS_OFF_OS: When set, BTS or BTM is skipped if CPL = 0.	06_0FH
10	BTS_OFF_USR: When set, BTS or BTM is skipped if CPL > 0.	06_0FH

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
11	FREEZE_LBRS_ON_PMI: When set, the LBR stack is frozen on a PMI request.	If CPUID.01H:ECX[15] = 1 && CPUID.0AH:EAX[7:0] > 1
12	FREEZE_PERFMON_ON_PMI: When set, each ENABLE bit of the global counter control MSR are frozen (address 38FH) on a PMI request.	If CPUID.01H:ECX[15] = 1 && CPUID.0AH:EAX[7:0] > 1
13	ENABLE_UNCORE_PMI: When set, enables the logical processor to receive and generate PMI on behalf of the uncore.	06_1AH
14	FREEZE_WHILE_SMM: When set, freezes PerfMon and trace messages while in SMM.	If IA32_PERF_CAPABILITIES[12] = 1
15	RTM_DEBUG: When set, enables DR7 debug bit on XBEGIN.	If (CPUID.07H.00H:EBX[11] = 1)
63:16	Reserved.	
Register Address: 1DDH, 477		IA32_LER_FROM_IP
Last Event Record Source IP Register (R/W)		
63:0	FROM_IP The source IP of the recorded branch or event, in canonical form.	Reset Value: 0
Register Address: 1DEH, 478		IA32_LER_TO_IP
Last Event Record Destination IP Register (R/W)		
63:0	TO_IP The destination IP of the recorded branch or event, in canonical form.	Reset Value: 0
Register Address: 1E0H, 480		IA32_LER_INFO
Last Event Record Info Register (R/W)		
55:0	Undefined, may be zero or non-zero. Writes of non-zero values do not fault, but reads may return a different value.	Reset Value: 0
59:56	BR_TYPE The branch type recorded by this LBR. Encodings match those of IA32_LBR_X_INFO.	Reset Value: 0
60	Undefined, may be zero or non-zero. Writes of non-zero values do not fault, but reads may return a different value.	Reset Value: 0
61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel® TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
62	IN_TSX This LBR record records a branch that retired during a TSX transaction. On processors that do not support Intel® TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
63	MISPRED The recorded branch taken/not-taken resolution (for conditional branches) or target (for any indirect branch, including RETs) was mispredicted.	Reset Value: 0
Register Address: 1F2H, 498		IA32_SMRR_PHYSBASE
SMRR Base Address (Writeable only in SMM) Base address of SMM memory range.		If IA32_MTRRCAP.SMRR[11] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
7:0	Type. Specifies memory type of the range.	
11:8	Reserved.	
31:12	PhysBase SMRR physical Base Address.	
63:32	Reserved.	
Register Address: 1F3H, 499		IA32_SMRR_PHYSMASK
SMRR Range Mask (Writeable only in SMM) Range Mask of SMM memory range.		If IA32_MTRRCAP[SMRR] = 1
10:0	Reserved.	
11	Valid Enable range mask.	
31:12	PhysMask SMRR address range mask.	
63:32	Reserved.	
Register Address: 1F8H, 504		IA32_PLATFORM_DCA_CAP
DCA Capability (R)		If CPUID.01H:ECX[18] = 1
Register Address: 1F9H, 505		IA32_CPU_DCA_CAP
If set, CPU supports Prefetch-Hint type.		If CPUID.01H:ECX[18] = 1
Register Address: 1FAH, 506		IA32_DCA_0_CAP
DCA type 0 Status and Control register.		If CPUID.01H:ECX[18] = 1
0	DCA_ACTIVE: Set by HW when DCA is fuse-enabled and no defeatures are set.	
2:1	TRANSACTION	
6:3	DCA_TYPE	
10:7	DCA_QUEUE_SIZE	
12:11	Reserved.	
16:13	DCA_DELAY: Writes will update the register but have no HW side-effect.	
23:17	Reserved.	
24	SW_BLOCK: SW can request DCA block by setting this bit.	
25	Reserved.	
26	HW_BLOCK: Set when DCA is blocked by HW (e.g., CRO.CD = 1).	
31:27	Reserved.	
Register Address: 200H, 512		IA32_MTRR_PHYSBASE0 (MTRRphysBase0)
See Section 14.11.2.3, "Variable Range MTRRs."		If IA32_MTRRCAP[7:0] > 0
Register Address: 201H, 513		IA32_MTRR_PHYSMASK0
MTRRphysMask0		If IA32_MTRRCAP[7:0] > 0
Register Address: 202H, 514		IA32_MTRR_PHYSBASE1
MTRRphysBase1		If IA32_MTRRCAP[7:0] > 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
MTRRphysMask1		If IA32_MTRRCAP[7:0] > 1
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
MTRRphysBase2		If IA32_MTRRCAP[7:0] > 2
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
MTRRphysMask2		If IA32_MTRRCAP[7:0] > 2
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
MTRRphysBase3		If IA32_MTRRCAP[7:0] > 3
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
MTRRphysMask3		If IA32_MTRRCAP[7:0] > 3
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
MTRRphysBase4		If IA32_MTRRCAP[7:0] > 4
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
MTRRphysMask4		If IA32_MTRRCAP[7:0] > 4
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
MTRRphysBase5		If IA32_MTRRCAP[7:0] > 5
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
MTRRphysMask5		If IA32_MTRRCAP[7:0] > 5
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
MTRRphysBase6		If IA32_MTRRCAP[7:0] > 6
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
MTRRphysMask6		If IA32_MTRRCAP[7:0] > 6
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
MTRRphysBase7		If IA32_MTRRCAP[7:0] > 7
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
MTRRphysMask7		If IA32_MTRRCAP[7:0] > 7
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
MTRRphysBase8		If IA32_MTRRCAP[7:0] > 8
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
MTRRphysMask8		If IA32_MTRRCAP[7:0] > 8
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
MTRRphysBase9		If IA32_MTRRCAP[7:0] > 9
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
MTRRphysMask9		If IA32_MTRRCAP[7:0] > 9
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
MTRRfix64K_00000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MTRRfix16K_80000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 259H, 601		IA32_MTRR_FIX16K_A0000
MTRRfix16K_A0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 268H, 616		IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000)
See Section 14.11.2.2, “Fixed Range MTRRs.”		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 269H, 617		IA32_MTRR_FIX4K_C8000
MTRRfix4K_C8000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26AH, 618		IA32_MTRR_FIX4K_D0000
MTRRfix4K_D0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26BH, 619		IA32_MTRR_FIX4K_D8000
MTRRfix4K_D8000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26CH, 620		IA32_MTRR_FIX4K_E0000
MTRRfix4K_E0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26DH, 621		IA32_MTRR_FIX4K_E8000
MTRRfix4K_E8000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26EH, 622		IA32_MTRR_FIX4K_F0000
MTRRfix4K_F0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26FH, 623		IA32_MTRR_FIX4K_F8000
MTRRfix4K_F8000.		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 277H, 631		IA32_PAT
IA32_PAT (R/W)		If CPUID.01H:EDX.PAT[16] = 1
2:0	PA0	
7:3	Reserved.	
10:8	PA1	
15:11	Reserved.	
18:16	PA2	
23:19	Reserved.	
26:24	PA3	
31:27	Reserved.	
34:32	PA4	
39:35	Reserved.	
42:40	PA5	
47:43	Reserved.	
50:48	PA6	
55:51	Reserved.	
58:56	PA7	
63:59	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 280H, 640		IA32_MCO_CTL2
MSR to enable/disable CMCI capability for bank 0. (R/W) See Section 18.3.2.5, "IA32_MCi_CTL2 MSRs."		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 0
14:0	Corrected error count threshold.	
29:15	Reserved.	
30	CMCI_EN	
63:31	Reserved.	
Register Address: 281H, 641		IA32_MC1_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 1
Register Address: 282H, 642		IA32_MC2_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 2
Register Address: 283H, 643		IA32_MC3_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 3
Register Address: 284H, 644		IA32_MC4_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 4
Register Address: 285H, 645		IA32_MC5_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 5
Register Address: 286H, 646		IA32_MC6_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 6
Register Address: 287H, 647		IA32_MC7_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 7
Register Address: 288H, 648		IA32_MC8_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 8
Register Address: 289H, 649		IA32_MC9_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 9
Register Address: 28AH, 650		IA32_MC10_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 10
Register Address: 28BH, 651		IA32_MC11_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 11
Register Address: 28CH, 652		IA32_MC12_CTL2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 12
Register Address: 28DH, 653	IA32_MC13_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 13
Register Address: 28EH, 654	IA32_MC14_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 14
Register Address: 28FH, 655	IA32_MC15_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 15
Register Address: 290H, 656	IA32_MC16_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 16
Register Address: 291H, 657	IA32_MC17_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 17
Register Address: 292H, 658	IA32_MC18_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 18
Register Address: 293H, 659	IA32_MC19_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 19
Register Address: 294H, 660	IA32_MC20_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 20
Register Address: 295H, 661	IA32_MC21_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 21
Register Address: 296H, 662	IA32_MC22_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 22
Register Address: 297H, 663	IA32_MC23_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 23
Register Address: 298H, 664	IA32_MC24_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 24
Register Address: 299H, 665	IA32_MC25_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 25
Register Address: 29AH, 666	IA32_MC26_CTL2	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 26
Register Address: 29BH, 667		IA32_MC27_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 27
Register Address: 29CH, 668		IA32_MC28_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 28
Register Address: 29DH, 669		IA32_MC29_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 29
Register Address: 29EH, 670		IA32_MC30_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 30
Register Address: 29FH, 671		IA32_MC31_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 31
Register Address: 2DCH, 732		IA32_INTEGRITY_STATUS
IA32_INTEGRITY_STATUS (R/O) Provides status information for integrity features.		If any one enumeration condition for defined bit field holds.
0	STATIC_LOCKSTEP 0: Static Lockstep Mode is not active on this logical processor. 1: Static Lockstep Mode is active on this logical processor.	If CPUID.07H.01H:EDX[24] = 1
63:1	Reserved.	
Register Address: 2FFH, 767		IA32_MTRR_DEF_TYPE
MTRRdefType (R/W)		If CPUID.01H:EDX.MTRR[12] = 1
2:0	Default Memory Type	
9:3	Reserved.	
10	Fixed Range MTRR Enable	
11	MTRR Enable	
63:12	Reserved.	
Register Address: 309H, 777		IA32_FIXED_CTR0
Fixed-Function Performance Counter 0 (R/W): Counts Instr_Retired.Any.		If CPUID.0AH:EDX[4:0] > 0 CPUID.0AH:ECX[0] = 1 CPUID.23H.01H:EBX[0] = 1
Register Address: 30AH, 778		IA32_FIXED_CTR1
Fixed-Function Performance Counter 1 (R/W): Counts CPU_CLK_Unhalted.Core.		If CPUID.0AH:EDX[4:0] > 1 CPUID.0AH:ECX[1] = 1 CPUID.23H.01H:EBX[1] = 1
Register Address: 30BH, 779		IA32_FIXED_CTR2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Fixed-Function Performance Counter 2 (R/W): Counts CPU_CLK_Unhalted.Ref.		If CPUID.0AH:EDX[4:0] > 2 CPUID.0AH:ECX[2] = 1 CPUID.23H.01H:EBX[2] = 1
Register Address: 30CH, 780		IA32_FIXED_CTR3
Fixed-Function Performance Counter 3 (R/W): Top-down Microarchitecture Analysis unhalted number of available slots.		If CPUID.0AH:EDX[4:0] > 3 CPUID.0AH:ECX[3] = 1 CPUID.23H.01H:EBX[3] = 1
Register Address: 30DH, 781		IA32_FIXED_CTR4
Fixed-Function Performance Counter 4 (R/W): Top-down bad speculation.		If CPUID.0AH:EDX[4:0] > 4 CPUID.0AH:ECX[4] = 1 CPUID.23H.01H:EBX[4] = 1
47:0	FIXED_COUNTER Top-down bad speculation counter.	
63:46	Reserved.	
Register Address: 30EH, 782		IA32_FIXED_CTR5
Fixed-Function Performance Counter 5 (R/W): Top-down Frontend Bound.		If CPUID.0AH:EDX[4:0] > 5 CPUID.0AH:ECX[5] = 1 CPUID.23H.01H:EBX[5] = 1
47:0	FIXED_COUNTER Top-down Frontend Bound counter.	
63:46	Reserved.	
Register Address: 30FH, 783		IA32_FIXED_CTR6
Fixed-Function Performance Counter 6 (R/W): Top-down retiring.		If CPUID.0AH:EDX[4:0] > 6 CPUID.0AH:ECX[6] = 1 CPUID.23H.01H:EBX[6] = 1
47:0	FIXED_COUNTER Top-down Retiring counter.	
63:46	Reserved.	
Register Address: 345H, 837		IA32_PERF_CAPABILITIES
Read Only MSR that enumerates the existence of performance monitoring features. (R/O)		If CPUID.01H:ECX[15] = 1
5:0	LBR format	
6	PEBS Trap	
7	PEBSSaveArchRegs	
11:8	PEBS Record Format	
12	1: Freeze while SMM is supported.	
13	1: Full width of counter writable via IA32_A_PMCx.	
14	PEBS_BASELINE	
15	1: Performance metrics available.	
16	1: PEBS output will be written into the Intel PT trace stream.	If CPUID.07H.00H:EBX[25] = 1
17	1: Indicates support for PEBS Retire Latency output.	
18	TSX_ADDRESS	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
19	RDPMC_METRICS_CLEAR	
63:20	Reserved.	
Register Address: 38DH, 909		IA32_FIXED_CTR_CTRL
Fixed-Function Performance Counter Control (R/W) Counter increments while the results of ANDing respective enable bit in IA32_PERF_GLOBAL_CTRL with the corresponding OS or USR bits in this MSR is true.		If CPUID.0AH:EAX[7:0] > 1
0	EN0_OS: Enable Fixed Counter 0 to count while CPL = 0.	
1	EN0_Usr: Enable Fixed Counter 0 to count while CPL > 0.	
2	AnyThr0: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15] = 0
3	EN0_PMI: Enable PMI when fixed counter 0 overflows.	
4	EN1_OS: Enable Fixed Counter 1 to count while CPL = 0.	
5	EN1_Usr: Enable Fixed Counter 1 to count while CPL > 0.	
6	AnyThr1: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15] = 0
7	EN1_PMI: Enable PMI when fixed counter 1 overflows.	
8	EN2_OS: Enable Fixed Counter 2 to count while CPL = 0.	
9	EN2_Usr: Enable Fixed Counter 2 to count while CPL > 0.	
10	AnyThr2: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15] = 0
11	EN2_PMI: Enable PMI when fixed counter 2 overflows.	
12	EN3_OS: Enable Fixed Counter 3 to count while CPL = 0.	
13	EN3_Usr: Enable Fixed Counter 3 to count while CPL > 0.	
14	Reserved.	
15	EN3_PMI: Enable PMI when fixed counter 3 overflows.	
63:16	Reserved.	
Register Address: 38EH, 910		IA32_PERF_GLOBAL_STATUS
Global Performance Counter Status (R/O)		If CPUID.0AH:EAX[7:0] > 0 (CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1)
0	Ovf_PMC0: Overflow status of IA32_PMC0.	If CPUID.0AH:EAX[15:8] > 0
1	Ovf_PMC1: Overflow status of IA32_PMC1.	If CPUID.0AH:EAX[15:8] > 1
2	Ovf_PMC2: Overflow status of IA32_PMC2.	If CPUID.0AH:EAX[15:8] > 2
3	Ovf_PMC3: Overflow status of IA32_PMC3.	If CPUID.0AH:EAX[15:8] > 3
n	Ovf_PMCn: Overflow status of IA32_PMCn.	If CPUID.0AH:EAX[15:8] > n

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
31:n+1	Reserved.	
32	Ovf_FixedCtr0: Overflow status of IA32_FIXED_CTR0.	If CPUID.0AH:EAX[7:0] > 1
33	Ovf_FixedCtr1: Overflow status of IA32_FIXED_CTR1.	If CPUID.0AH:EAX[7:0] > 1
34	Ovf_FixedCtr2: Overflow status of IA32_FIXED_CTR2.	If CPUID.0AH:EAX[7:0] > 1
32+m	Ovf_FixedCtrm: Overflow status of IA32_FIXED_CTRm.	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m
47:33+m	Reserved.	
48	OVF_PERF_METRICS: If this bit is set, it indicates that PERF_METRIC counter has overflowed and a PMI is triggered; however, an overflow of fixed counter 3 should normally happen first. If this bit is clear no overflow occurred.	
54:49	Reserved.	
55	Trace_ToPA_PMI: A PMI occurred due to a ToPA entry memory buffer that was completely filled.	If CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1
57:56	Reserved.	
58	LBR_Frz. LBRs are frozen due to: - IA32_DEBUGCTL.FREEZE_LBR_ON_PMI=1. - The LBR stack overflowed.	If CPUID.0AH:EAX[7:0] > 3
59	CTR_Frz. Performance counters in the core PMU are frozen due to: - IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI=1. - One or more core PMU counters overflowed.	If CPUID.0AH:EAX[7:0] > 3
60	ASCI: Data in the performance counters in the core PMU may include contributions from the direct or indirect operation Intel SGX to protect an enclave.	If the processor supports Intel® SGX.
61	Ovf_Uncore: Uncore counter overflow status.	If CPUID.0AH:EAX[7:0] > 2
62	OvfBuf: DS SAVE area Buffer overflow status.	If CPUID.0AH:EAX[7:0] > 0
63	CondChgd: Status bits of this register have changed.	If CPUID.0AH:EAX[7:0] > 0
Register Address: 38FH, 911		IA32_PERF_GLOBAL_CTRL
Global Performance Counter Control (R/W) Counter increments while the result of ANDing the respective enable bit in this MSR with the corresponding OS or USR bits in the general-purpose or fixed counter control MSR is true.		If CPUID.0AH:EAX[7:0] > 0
0	EN_PMC0	If CPUID.0AH:EAX[15:8] > 0
1	EN_PMC1	If CPUID.0AH:EAX[15:8] > 1
2	EN_PMC2	If CPUID.0AH:EAX[15:8] > 2
n	EN_PMCn	If CPUID.0AH:EAX[15:8] > n
31:n+1	Reserved.	
32	EN_FIXED_CTR0	If CPUID.0AH:EDX[4:0] > 0
33	EN_FIXED_CTR1	If CPUID.0AH:EDX[4:0] > 1
34	EN_FIXED_CTR2	If CPUID.0AH:EDX[4:0] > 2
32+m	EN_FIXED_CTRm	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
47:33+m	Reserved.	
48	EN_PERF_METRICS: If this bit is set and fixed counter 3 is effectively enabled, built-in performance metrics are enabled.	
63:49	Reserved.	
Register Address: 390H, 912		IA32_PERF_GLOBAL_STATUS_RESET
Global Performance Counter Overflow Reset Control (R/W)		If CPUID.0AH:EAX[7:0] > 3 (CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1)
0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH:EAX[15:8] > 0
1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH:EAX[15:8] > 1
2	Set 1 to Clear Ovf_PMC2 bit.	If CPUID.0AH:EAX[15:8] > 2
n	Set 1 to Clear Ovf_PMCn bit.	If CPUID.0AH:EAX[15:8] > n
31:n	Reserved.	
32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.0AH:EDX[4:0] > 0
33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH:EDX[4:0] > 1
34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH:EDX[4:0] > 2
32+m	Set 1 to Clear Ovf_FIXED_CTRm bit.	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m
47:33+m	Reserved.	
48	RESET_OVF_PERF_METRICS: If this bit is set, it will clear the status bit in the IA32_PERF_GLOBAL_STATUS register for the PERF_METRICS counters.	
54:49	Reserved.	
55	Set 1 to Clear Trace_ToPA_PMI bit.	If CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1
57:56	Reserved.	
58	Set 1 to Clear LBR_Frz bit.	If CPUID.0AH:EAX[7:0] > 3
59	Set 1 to Clear CTR_Frz bit.	If CPUID.0AH:EAX[7:0] > 3
60	Set 1 to Clear ASCII bit.	If the processor supports Intel® SGX.
61	Set 1 to Clear Ovf_Uncore bit.	06_2EH
62	Set 1 to Clear OvfBuf bit.	If CPUID.0AH:EAX[7:0] > 0
63	Set 1 to clear CondChgd bit.	If CPUID.0AH:EAX[7:0] > 0
Register Address: 391H, 913		IA32_PERF_GLOBAL_STATUS_SET
Global Performance Counter Overflow Set Control (R/W)		If CPUID.0AH:EAX[7:0] > 3 (CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1)
0	Set 1 to cause Ovf_PMC0 = 1.	If CPUID.0AH:EAX[7:0] > 3
1	Set 1 to cause Ovf_PMC1 = 1.	If CPUID.0AH:EAX[15:8] > 1
2	Set 1 to cause Ovf_PMC2 = 1.	If CPUID.0AH:EAX[15:8] > 2
n	Set 1 to cause Ovf_PMCn = 1.	If CPUID.0AH:EAX[15:8] > n

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
31:n	Reserved.	
32	Set 1 to cause Ovf_FIXED_CTR0 = 1.	If CPUID.0AH:EAX[7:0] > 3
33	Set 1 to cause Ovf_FIXED_CTR1 = 1.	If CPUID.0AH:EAX[7:0] > 3
34	Set 1 to cause Ovf_FIXED_CTR2 = 1.	If CPUID.0AH:EAX[7:0] > 3
32+m	Set 1 to cause Ovf_FIXED_CTRm = 1.	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m
47:33+m	Reserved.	
48	SET_OVF_PERF_METRICS: If this bit is set, it will set the status bit in the IA32_PERF_GLOBAL_STATUS register for the PERF_METRICS counters.	
54:49	Reserved.	
55	Set 1 to cause Trace_ToPA_PMI = 1.	If CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1
57:56	Reserved.	
58	Set 1 to cause LBR_Frз = 1.	If CPUID.0AH:EAX[7:0] > 3
59	Set 1 to cause CTR_Frз = 1.	If CPUID.0AH:EAX[7:0] > 3
60	Set 1 to cause ASCI = 1.	If the processor supports Intel® SGX.
61	Set 1 to cause Ovf_Uncore = 1.	If CPUID.0AH:EAX[7:0] > 3
62	Set 1 to cause OvfBuf = 1.	If CPUID.0AH:EAX[7:0] > 3
63	Reserved.	
Register Address: 392H, 914		IA32_PERF_GLOBAL_INUSE
Indicator that core PerfMon interface is in use. (R/O)		If CPUID.0AH:EAX[7:0] > 3
0	IA32_PERFEVTSEL0 in use.	
1	IA32_PERFEVTSEL1 in use.	If CPUID.0AH:EAX[15:8] > 1
2	IA32_PERFEVTSEL2 in use.	If CPUID.0AH:EAX[15:8] > 2
n	IA32_PERFEVTSELn in use.	If CPUID.0AH:EAX[15:8] > n
31:n+1	Reserved.	
32	IA32_FIXED_CTR0 in use.	
33	IA32_FIXED_CTR1 in use.	
34	IA32_FIXED_CTR2 in use.	
32+m	IA32_FIXED_CTRm in use.	
62:33+m	Reserved or model specific.	
63	PMI in use.	
Register Address: 3F1H, 1009		IA32_PEBBS_ENABLE
PEBS Control (R/W)		
0	Enable PEBS on IA32_PMC0.	06_0FH
3:1	Reserved or model specific.	
31:4	Reserved.	
35:32	Reserved or model specific.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:36	Reserved.	
Register Address: 3F4H, 1012		IA32_PEBS_BASE
Architectural PEBS Output Buffer Base Register (R/W) Holds basic structural controls for Architectural PEBS. This MSR is initialized to all zeros on reset.		CPUID.23H.00H:EAX[5]=1
3:0	SIZE (R/W) Size options for the PEBS output buffer (actual size is power-of-2 multiples of 4KB, that is, Buffer Size = 4KB * 2 ^{SIZE}).	
11:4	Reserved.	
MAXPHYADDR - 1:12	PHYS_ADDR (R/W) Start of the PEBS output buffer (4KB-aligned physical address, excluding the 12b page offset).	
63:MAXPHYADDR	Reserved.	
Register Address: 3F5H, 1013		IA32_PEBS_INDEX
Arch PEBS Output Buffer Index Register (R/W) Holds status and additional controls for Architectural PEBS. This MSR is initialized to all zeros on reset.		CPUID.23H.00H:EAX[5]=1
3:0	Reserved.	
26:4	WR_OFFSET (R/W) Next-byte-to-write offset from IA32_PEBS_BASE.PHYS_ADDR.	
30:27	Reserved.	
31	FULL (R/W) The buffer is full, and no further records will be generated.	
32	THRESH_EN (R/W) Enable PMI on PEBS buffer Threshold checks.	
35:33	Reserved.	
58:36	THRESH_OFFSET (R/W) The threshold for the PEBS buffer (specified as offset in bytes from IA32_PEBS_BASE.PHYS_ADDR).	
63:59	Reserved.	
Register Address: 400H, 1024		IA32_MCO_CTL
MCO_CTL		If IA32_MCG_CAP.CNT > 0
Register Address: 401H, 1025		IA32_MCO_STATUS
MCO_STATUS		If IA32_MCG_CAP.CNT > 0
Register Address: 402H, 1026		IA32_MCO_ADDR ¹
MCO_ADDR		If IA32_MCG_CAP.CNT > 0
Register Address: 403H, 1027		IA32_MCO_MISC
MCO_MISC		If IA32_MCG_CAP.CNT > 0
Register Address: 404H, 1028		IA32_MC1_CTL
MC1_CTL		If IA32_MCG_CAP.CNT > 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: 405H, 1029		IA32_MC1_STATUS	
MC1_STATUS		If IA32_MCG_CAP.CNT > 1	
Register Address: 406H, 1030		IA32_MC1_ADDR ²	
MC1_ADDR		If IA32_MCG_CAP.CNT > 1	
Register Address: 407H, 1031		IA32_MC1_MISC	
MC1_MISC		If IA32_MCG_CAP.CNT > 1	
Register Address: 408H, 1032		IA32_MC2_CTL	
MC2_CTL		If IA32_MCG_CAP.CNT > 2	
Register Address: 409H, 1033		IA32_MC2_STATUS	
MC2_STATUS		If IA32_MCG_CAP.CNT > 2	
Register Address: 40AH, 1034		IA32_MC2_ADDR ¹	
MC2_ADDR		If IA32_MCG_CAP.CNT > 2	
Register Address: 40BH, 1035		IA32_MC2_MISC	
MC2_MISC		If IA32_MCG_CAP.CNT > 2	
Register Address: 40CH, 1036		IA32_MC3_CTL	
MC3_CTL		If IA32_MCG_CAP.CNT > 3	
Register Address: 40DH, 1037		IA32_MC3_STATUS	
MC3_STATUS		If IA32_MCG_CAP.CNT > 3	
Register Address: 40EH, 1038		IA32_MC3_ADDR ¹	
MC3_ADDR		If IA32_MCG_CAP.CNT > 3	
Register Address: 40FH, 1039		IA32_MC3_MISC	
MC3_MISC		If IA32_MCG_CAP.CNT > 3	
Register Address: 410H, 1040		IA32_MC4_CTL	
MC4_CTL		If IA32_MCG_CAP.CNT > 4	
Register Address: 411H, 1041		IA32_MC4_STATUS	
MC4_STATUS		If IA32_MCG_CAP.CNT > 4	
Register Address: 412H, 1042		IA32_MC4_ADDR ¹	
MC4_ADDR		If IA32_MCG_CAP.CNT > 4	
Register Address: 413H, 1043		IA32_MC4_MISC	
MC4_MISC		If IA32_MCG_CAP.CNT > 4	
Register Address: 414H, 1044		IA32_MC5_CTL	
MC5_CTL		If IA32_MCG_CAP.CNT > 5	
Register Address: 415H, 1045		IA32_MC5_STATUS	
MC5_STATUS		If IA32_MCG_CAP.CNT > 5	
Register Address: 416H, 1046		IA32_MC5_ADDR ¹	
MC5_ADDR		If IA32_MCG_CAP.CNT > 5	
Register Address: 417H, 1047		IA32_MC5_MISC	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MC5_MISC		If IA32_MCG_CAP.CNT > 5
Register Address: 418H, 1048	IA32_MC6_CTL	
MC6_CTL		If IA32_MCG_CAP.CNT > 6
Register Address: 419H, 1049	IA32_MC6_STATUS	
MC6_STATUS		If IA32_MCG_CAP.CNT > 6
Register Address: 41AH, 1050	IA32_MC6_ADDR ¹	
MC6_ADDR		If IA32_MCG_>6CAP.CNT > 6
Register Address: 41BH, 1051	IA32_MC6_MISC	
MC6_MISC		If IA32_MCG_CAP.CNT > 6
Register Address: 41CH, 1052	IA32_MC7_CTL	
MC7_CTL		If IA32_MCG_CAP.CNT > 7
Register Address: 41DH, 1053	IA32_MC7_STATUS	
MC7_STATUS		If IA32_MCG_CAP.CNT > 7
Register Address: 41EH, 1054	IA32_MC7_ADDR ¹	
MC7_ADDR		If IA32_MCG_CAP.CNT > 7
Register Address: 41FH, 1055	IA32_MC7_MISC	
MC7_MISC		If IA32_MCG_CAP.CNT > 7
Register Address: 420H, 1056	IA32_MC8_CTL	
MC8_CTL		If IA32_MCG_CAP.CNT > 8
Register Address: 421H, 1057	IA32_MC8_STATUS	
MC8_STATUS		If IA32_MCG_CAP.CNT > 8
Register Address: 422H, 1058	IA32_MC8_ADDR ¹	
MC8_ADDR		If IA32_MCG_CAP.CNT > 8
Register Address: 423H, 1059	IA32_MC8_MISC	
MC8_MISC		If IA32_MCG_CAP.CNT > 8
Register Address: 424H, 1060	IA32_MC9_CTL	
MC9_CTL		If IA32_MCG_CAP.CNT > 9
Register Address: 425H, 1061	IA32_MC9_STATUS	
MC9_STATUS		If IA32_MCG_CAP.CNT > 9
Register Address: 426H, 1062	IA32_MC9_ADDR ¹	
MC9_ADDR		If IA32_MCG_CAP.CNT > 9
Register Address: 427H, 1063	IA32_MC9_MISC	
MC9_MISC		If IA32_MCG_CAP.CNT > 9
Register Address: 428H, 1064	IA32_MC10_CTL	
MC10_CTL		If IA32_MCG_CAP.CNT > 10
Register Address: 429H, 1065	IA32_MC10_STATUS	
MC10_STATUS		If IA32_MCG_CAP.CNT > 10

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: 42AH, 1066		IA32_MC10_ADDR ¹	
MC10_ADDR		If IA32_MCG_CAP.CNT > 10	
Register Address: 42BH, 1067		IA32_MC10_MISC	
MC10_MISC		If IA32_MCG_CAP.CNT > 10	
Register Address: 42CH, 1068		IA32_MC11_CTL	
MC11_CTL		If IA32_MCG_CAP.CNT > 11	
Register Address: 42DH, 1069		IA32_MC11_STATUS	
MC11_STATUS		If IA32_MCG_CAP.CNT > 11	
Register Address: 42EH, 1070		IA32_MC11_ADDR ¹	
MC11_ADDR		If IA32_MCG_CAP.CNT > 11	
Register Address: 42FH, 1071		IA32_MC11_MISC	
MC11_MISC		If IA32_MCG_CAP.CNT > 11	
Register Address: 430H, 1072		IA32_MC12_CTL	
MC12_CTL		If IA32_MCG_CAP.CNT > 12	
Register Address: 431H, 1073		IA32_MC12_STATUS	
MC12_STATUS		If IA32_MCG_CAP.CNT > 12	
Register Address: 432H, 1074		IA32_MC12_ADDR ¹	
MC12_ADDR		If IA32_MCG_CAP.CNT > 12	
Register Address: 433H, 1075		IA32_MC12_MISC	
MC12_MISC		If IA32_MCG_CAP.CNT > 12	
Register Address: 434H, 1076		IA32_MC13_CTL	
MC13_CTL		If IA32_MCG_CAP.CNT > 13	
Register Address: 435H, 1077		IA32_MC13_STATUS	
MC13_STATUS		If IA32_MCG_CAP.CNT > 13	
Register Address: 436H, 1078		IA32_MC13_ADDR ¹	
MC13_ADDR		If IA32_MCG_CAP.CNT > 13	
Register Address: 437H, 1079		IA32_MC13_MISC	
MC13_MISC		If IA32_MCG_CAP.CNT > 13	
Register Address: 438H, 1080		IA32_MC14_CTL	
MC14_CTL		If IA32_MCG_CAP.CNT > 14	
Register Address: 439H, 1081		IA32_MC14_STATUS	
MC14_STATUS		If IA32_MCG_CAP.CNT > 14	
Register Address: 43AH, 1082		IA32_MC14_ADDR ¹	
MC14_ADDR		If IA32_MCG_CAP.CNT > 14	
Register Address: 43BH, 1083		IA32_MC14_MISC	
MC14_MISC		If IA32_MCG_CAP.CNT > 14	
Register Address: 43CH, 1084		IA32_MC15_CTL	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MC15_CTL		If IA32_MCG_CAP.CNT > 15
Register Address: 43DH, 1085	IA32_MC15_STATUS	
MC15_STATUS		If IA32_MCG_CAP.CNT > 15
Register Address: 43EH, 1086	IA32_MC15_ADDR ¹	
MC15_ADDR		If IA32_MCG_CAP.CNT > 15
Register Address: 43FH, 1087	IA32_MC15_MISC	
MC15_MISC		If IA32_MCG_CAP.CNT > 15
Register Address: 440H, 1088	IA32_MC16_CTL	
MC16_CTL		If IA32_MCG_CAP.CNT > 16
Register Address: 441H, 1089	IA32_MC16_STATUS	
MC16_STATUS		If IA32_MCG_CAP.CNT > 16
Register Address: 442H, 1090	IA32_MC16_ADDR ¹	
MC16_ADDR		If IA32_MCG_CAP.CNT > 16
Register Address: 443H, 1091	IA32_MC16_MISC	
MC16_MISC		If IA32_MCG_CAP.CNT > 16
Register Address: 444H, 1092	IA32_MC17_CTL	
MC17_CTL		If IA32_MCG_CAP.CNT > 17
Register Address: 445H, 1093	IA32_MC17_STATUS	
MC17_STATUS		If IA32_MCG_CAP.CNT > 17
Register Address: 446H, 1094	IA32_MC17_ADDR ¹	
MC17_ADDR		If IA32_MCG_CAP.CNT > 17
Register Address: 447H, 1095	IA32_MC17_MISC	
MC17_MISC		If IA32_MCG_CAP.CNT > 17
Register Address: 448H, 1096	IA32_MC18_CTL	
MC18_CTL		If IA32_MCG_CAP.CNT > 18
Register Address: 449H, 1097	IA32_MC18_STATUS	
MC18_STATUS		If IA32_MCG_CAP.CNT > 18
Register Address: 44AH, 1098	IA32_MC18_ADDR ¹	
MC18_ADDR		If IA32_MCG_CAP.CNT > 18
Register Address: 44BH, 1099	IA32_MC18_MISC	
MC18_MISC		If IA32_MCG_CAP.CNT > 18
Register Address: 44CH, 1100	IA32_MC19_CTL	
MC19_CTL		If IA32_MCG_CAP.CNT > 19
Register Address: 44DH, 1101	IA32_MC19_STATUS	
MC19_STATUS		If IA32_MCG_CAP.CNT > 19
Register Address: 44EH, 1102	IA32_MC19_ADDR ¹	
MC19_ADDR		If IA32_MCG_CAP.CNT > 19

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: 44FH, 1103		IA32_MC19_MISC	
MC19_MISC		If IA32_MCG_CAP.CNT > 19	
Register Address: 450H, 1104		IA32_MC20_CTL	
MC20_CTL		If IA32_MCG_CAP.CNT > 20	
Register Address: 451H, 1105		IA32_MC20_STATUS	
MC20_STATUS		If IA32_MCG_CAP.CNT > 20	
Register Address: 452H, 1106		IA32_MC20_ADDR ¹	
MC20_ADDR		If IA32_MCG_CAP.CNT > 20	
Register Address: 453H, 1107		IA32_MC20_MISC	
MC20_MISC		If IA32_MCG_CAP.CNT > 20	
Register Address: 454H, 1108		IA32_MC21_CTL	
MC21_CTL		If IA32_MCG_CAP.CNT > 21	
Register Address: 455H, 1109		IA32_MC21_STATUS	
MC21_STATUS		If IA32_MCG_CAP.CNT > 21	
Register Address: 456H, 1110		IA32_MC21_ADDR ¹	
MC21_ADDR		If IA32_MCG_CAP.CNT > 21	
Register Address: 457H, 1111		IA32_MC21_MISC	
MC21_MISC		If IA32_MCG_CAP.CNT > 21	
Register Address: 458H, 1112		IA32_MC22_CTL	
MC22_CTL		If IA32_MCG_CAP.CNT > 22	
Register Address: 459H, 1113		IA32_MC22_STATUS	
MC22_STATUS		If IA32_MCG_CAP.CNT > 22	
Register Address: 45AH, 1114		IA32_MC22_ADDR ¹	
MC22_ADDR		If IA32_MCG_CAP.CNT > 22	
Register Address: 45BH, 1115		IA32_MC22_MISC	
MC22_MISC		If IA32_MCG_CAP.CNT > 22	
Register Address: 45CH, 1116		IA32_MC23_CTL	
MC23_CTL		If IA32_MCG_CAP.CNT > 23	
Register Address: 45DH, 1117		IA32_MC23_STATUS	
MC23_STATUS		If IA32_MCG_CAP.CNT > 23	
Register Address: 45EH, 1118		IA32_MC23_ADDR ¹	
MC23_ADDR		If IA32_MCG_CAP.CNT > 23	
Register Address: 45FH, 1119		IA32_MC23_MISC	
MC23_MISC		If IA32_MCG_CAP.CNT > 23	
Register Address: 460H, 1120		IA32_MC24_CTL	
MC24_CTL		If IA32_MCG_CAP.CNT > 24	
Register Address: 461H, 1121		IA32_MC24_STATUS	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MC24_STATUS		If IA32_MCG_CAP.CNT > 24
Register Address: 462H, 1122		IA32_MC24_ADDR ¹
MC24_ADDR		If IA32_MCG_CAP.CNT > 24
Register Address: 463H, 1123		IA32_MC24_MISC
MC24_MISC		If IA32_MCG_CAP.CNT > 24
Register Address: 464H, 1124		IA32_MC25_CTL
MC25_CTL		If IA32_MCG_CAP.CNT > 25
Register Address: 465H, 1125		IA32_MC25_STATUS
MC25_STATUS		If IA32_MCG_CAP.CNT > 25
Register Address: 466H, 1126		IA32_MC25_ADDR ¹
MC25_ADDR		If IA32_MCG_CAP.CNT > 25
Register Address: 467H, 1127		IA32_MC25_MISC
MC25_MISC		If IA32_MCG_CAP.CNT > 25
Register Address: 468H, 1128		IA32_MC26_CTL
MC26_CTL		If IA32_MCG_CAP.CNT > 26
Register Address: 469H, 1129		IA32_MC26_STATUS
MC26_STATUS		If IA32_MCG_CAP.CNT > 26
Register Address: 46AH, 1130		IA32_MC26_ADDR ¹
MC26_ADDR		If IA32_MCG_CAP.CNT > 26
Register Address: 46BH, 1131		IA32_MC26_MISC
MC26_MISC		If IA32_MCG_CAP.CNT > 26
Register Address: 46CH, 1132		IA32_MC27_CTL
MC27_CTL		If IA32_MCG_CAP.CNT > 27
Register Address: 46DH, 1133		IA32_MC27_STATUS
MC27_STATUS		If IA32_MCG_CAP.CNT > 27
Register Address: 46EH, 1134		IA32_MC27_ADDR ¹
MC27_ADDR		If IA32_MCG_CAP.CNT > 27
Register Address: 46FH, 1135		IA32_MC27_MISC
MC27_MISC		If IA32_MCG_CAP.CNT > 27
Register Address: 470H, 1136		IA32_MC28_CTL
MC28_CTL		If IA32_MCG_CAP.CNT > 28
Register Address: 471H, 1137		IA32_MC28_STATUS
MC28_STATUS		If IA32_MCG_CAP.CNT > 28
Register Address: 472H, 1138		IA32_MC28_ADDR ¹
MC28_ADDR		If IA32_MCG_CAP.CNT > 28
Register Address: 473H, 1139		IA32_MC28_MISC
MC28_MISC		If IA32_MCG_CAP.CNT > 28

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 474H, 1140	IA32_MC29_CTL	
MC29_CTL		If IA32_MCG_CAP.CNT > 29
Register Address: 475H, 1141	IA32_MC29_STATUS	
MC29_STATUS		If IA32_MCG_CAP.CNT > 29
Register Address: 476H, 1142	IA32_MC29_ADDR	
MC29_ADDR		If IA32_MCG_CAP.CNT > 29
Register Address: 477H, 1143	IA32_MC29_MISC	
MC29_MISC		If IA32_MCG_CAP.CNT > 29
Register Address: 478H, 1144	IA32_MC30_CTL	
MC30_CTL		If IA32_MCG_CAP.CNT > 30
Register Address: 479H, 1145	IA32_MC30_STATUS	
MC30_STATUS		If IA32_MCG_CAP.CNT > 30
Register Address: 47AH, 1146	IA32_MC30_ADDR	
MC30_ADDR		If IA32_MCG_CAP.CNT > 30
Register Address: 47BH, 1147	IA32_MC30_MISC	
MC30_MISC		If IA32_MCG_CAP.CNT > 30
Register Address: 47CH, 1148	IA32_MC31_CTL	
MC31_CTL		If IA32_MCG_CAP.CNT > 31
Register Address: 47DH, 1149	IA32_MC31_STATUS	
MC31_STATUS		If IA32_MCG_CAP.CNT > 31
Register Address: 47EH, 1150	IA32_MC31_ADDR	
MC31_ADDR		If IA32_MCG_CAP.CNT > 31
Register Address: 47FH, 1151	IA32_MC31_MISC	
MC31_MISC		If IA32_MCG_CAP.CNT > 31
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Appendix A.1, "Basic VMX Information."		If CPUID.01H:ECX[5] = 1
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3.1, "Pin-Based VM-Execution Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of Primary VM-Exit Controls (R/O) See Appendix A.4.1, "Primary VM-Exit Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 485H, 1157		IA32_VMX_MISC
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data."		If CPUID.01H:ECX[5] = 1
Register Address: 486H, 1158		IA32_VMX_CR0_FIXED0
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0."		If CPUID.01H:ECX[5] = 1
Register Address: 487H, 1159		IA32_VMX_CR0_FIXED1
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0."		If CPUID.01H:ECX[5] = 1
Register Address: 488H, 1160		IA32_VMX_CR4_FIXED0
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4."		If CPUID.01H:ECX[5] = 1
Register Address: 489H, 1161		IA32_VMX_CR4_FIXED1
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4."		If CPUID.01H:ECX[5] = 1
Register Address: 48AH, 1162		IA32_VMX_VMCS_ENUM
Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration."		If CPUID.01H:ECX[5] = 1
Register Address: 48BH, 1163		IA32_VMX_PROCBASED_CTL2
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.3, "Secondary Processor-Based VM-Execution Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTL2[63])
Register Address: 48CH, 1164		IA32_VMX_EPT_VPID_CAP
Capability Reporting Register of EPT and VPID (R/O) See Appendix A.10, "VPID and EPT Capabilities."		If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTL2[63 && (IA32_VMX_PROCBASED_CTL2[3 3] IA32_VMX_PROCBASED_CTL2[3 7]))
Register Address: 48DH, 1165		IA32_VMX_TRUE_PINBASED_CTL2
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Appendix A.3.1, "Pin-Based VM-Execution Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])
Register Address: 48EH, 1166		IA32_VMX_TRUE_PROCBASED_CTL2
Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])
Register Address: 48FH, 1167		IA32_VMX_TRUE_EXIT_CTL2
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Appendix A.4, "VM-Exit Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTL5	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Appendix A.5, "VM-Entry Controls."	If (CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])	
Register Address: 491H, 1169	IA32_VMX_VMFUNC	
Capability Reporting Register of VM-Function Controls (R/O)	If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTL5[63] && IA32_VMX_PROCBASED_CTL5[45])	
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTL53	
Capability Reporting Register of Tertiary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.4, "Tertiary Processor-Based VM-Execution Controls."	If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTL5[49])	
Register Address: 493H, 1171	IA32_VMX_EXIT_CTL52	
Capability Reporting Register of Secondary VM-Exit Controls (R/O) See Appendix A.4.2, "Secondary VM-Exit Controls."	If (CPUID.01H:ECX[5] && IA32_VMX_EXIT_CTL5[63])	
Register Address: 4C1H, 1217	IA32_A_PMC0	
Full Width Writable IA32_PMC0 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 0) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[0] = 1	
Register Address: 4C2H, 1218	IA32_A_PMC1	
Full Width Writable IA32_PMC1 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 1) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[1] = 1	
Register Address: 4C3H, 1219	IA32_A_PMC2	
Full Width Writable IA32_PMC2 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 2) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[2] = 1	
Register Address: 4C4H, 1220	IA32_A_PMC3	
Full Width Writable IA32_PMC3 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 3) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[3] = 1	
Register Address: 4C5H, 1221	IA32_A_PMC4	
Full Width Writable IA32_PMC4 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 4) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[4] = 1	
Register Address: 4C6H, 1222	IA32_A_PMC5	
Full Width Writable IA32_PMC5 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 5) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[5] = 1	
Register Address: 4C7H, 1223	IA32_A_PMC6	
Full Width Writable IA32_PMC6 Alias (R/W)	If (CPUID.0AH:EAX[15:8] > 6) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[6] = 1	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 4C8H, 1224		IA32_A_PMC7
Full Width Writable IA32_PMC7 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 7) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[7] = 1
Register Address: 4C9H, 1225		IA32_A_PMC8
Full Width Writable IA32_PMC8 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 8) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[8] = 1
Register Address: 4CAH, 1226		IA32_A_PMC9
Full Width Writable IA32_PMC9 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 9) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[9] = 1
Register Address: 4D0H, 1232		IA32_MCG_EXT_CTL
Allows software to signal some MCEs to only a single logical processor in the system. (R/W) See Section 18.3.1.4, "IA32_MCG_EXT_CTL MSR."		If IA32_MCG_CAP.LMCE_P = 1
0	LMCE_EN Enable / Disable local machine check exception.	
63:1	Reserved.	
Register Address: 500H, 1280		IA32_SGX_SVN_STATUS
Status and SVN Threshold of SGX Support for ACM (R/O)		If CPUID.07H.00H:EBX[2] = 1
0	Lock.	See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."
15:1	Reserved.	
23:16	SGX_SVN_SINIT	See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."
63:24	Reserved.	
Register Address: 560H, 1376		IA32_RTIT_OUTPUT_BASE
Trace Output Base Register (R/W)		If ((CPUID.07H.00H:EBX[25] = 1) && ((CPUID.14H.00H:ECX[0] = 1) (CPUID.14H.00H:ECX[2] = 1)))
6:0	Reserved.	
M-1:7	Base physical address.	M is the value enumerated by CPUID.80000008H:EAX[7:0].
63:M	Reserved.	
Register Address: 561H, 1377		IA32_RTIT_OUTPUT_MASK_PTRS
Trace Output Mask Pointers Register (R/W)		If ((CPUID.07H.00H:EBX[25] = 1) && ((CPUID.14H.00H:ECX[0] = 1) (CPUID.14H.00H:ECX[2] = 1)))
6:0	Reserved.	
31:7	MaskOrTableOffset.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:32	Output Offset.	
Register Address: 570H, 1392		IA32_RTIT_CTL
Trace Control Register (R/W)		If (CPUID.07H.00H:EBX[25] = 1)
0	TraceEn	
1	CYCEn	If (CPUID.07H.00H:EBX[1] = 1)
2	OS	
3	User	
4	PwrEvtEn	If (CPUID.07H.01H:EBX[5] = 1)
5	FUPonPTW	If (CPUID.07H.01H:EBX[4] = 1)
6	FabricEn	If (CPUID.07H.00H:ECX[3] = 1)
7	CR3Filter	If (CPUID.14H.00H:EBX[0] = 1)
8	ToPA	
9	MTCEn	If (CPUID.07H.00H:EBX[3] = 1)
10	TSCEn	
11	DisRETC	
12	PTWEn	If (CPUID.07H.01H:EBX[4] = 1)
13	BranchEn	
17:14	MTCFreq.	If (CPUID.07H.00H:EBX[3] = 1)
18	Reserved, must be zero.	
22:19	CycThresh	If (CPUID.07H.00H:EBX[1] = 1)
23	Reserved, must be zero.	
27:24	PSBFreq	If (CPUID.07H.00H:EBX[1] = 1)
30:28	Reserved, must be zero.	
31	EventEn	If (CPUID.14H.00H:EBX[7] = 1)
35:32	ADDR0_CFG	If (CPUID.07H.01H:EAX[2:0] > 0)
39:36	ADDR1_CFG	If (CPUID.07H.01H:EAX[2:0] > 1)
43:40	ADDR2_CFG	If (CPUID.07H.01H:EAX[2:0] > 2)
47:44	ADDR3_CFG	If (CPUID.07H.01H:EAX[2:0] > 3)
54:48	Reserved, must be zero.	
55	DisTNT	If (CPUID.14H.00H:EBX[8] = 1)
56	InjectPsbPmiOnEnable	If (CPUID.07H.01H:EBX[6] = 1)
63:57	Reserved, must be zero.	
Register Address: 571H, 1393		IA32_RTIT_STATUS
Tracing Status Register (R/W)		If (CPUID.07H.00H:EBX[25] = 1)
0	FilterEn (writes ignored).	If (CPUID.07H.00H:EBX[2] = 1)
1	ContexEn (writes ignored).	
2	TriggerEn (writes ignored).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
3	Reserved.	
4	Error	
5	Stopped	
6	PendPSB	If (CPUID.07H.00H:EBX[6] = 1)
7	PendToPAPMI	If (CPUID.07H.00H:EBX[6] = 1)
31:8	Reserved, must be zero.	
48:32	PacketByteCnt	If (CPUID.07H.00H:EBX[1] > 3)
63:49	Reserved.	
Register Address: 572H, 1394		IA32_RTIT_CR3_MATCH
Trace Filter CR3 Match Register (R/W)		If (CPUID.07H.00H:EBX[25] = 1)
4:0	Reserved.	
63:5	CR3[63:5] value to match.	
Register Address: 580H, 1408		IA32_RTIT_ADDR0_A
Region 0 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 0)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 581H, 1409		IA32_RTIT_ADDR0_B
Region 0 End Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 0)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 582H, 1410		IA32_RTIT_ADDR1_A
Region 1 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 1)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 583H, 1411		IA32_RTIT_ADDR1_B
Region 1 End Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 1)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 584H, 1412		IA32_RTIT_ADDR2_A
Region 2 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 2)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 585H, 1413		IA32_RTIT_ADDR2_B
Region 2 End Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 2)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 586H, 1414		IA32_RTIT_ADDR3_A

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Region 3 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 3)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 587H, 1415		IA32_RTIT_ADDR3_B
Region 3 End Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 3)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 600H, 1536		IA32_DS_AREA
DS Save Area (R/W) Points to the linear address of the first byte of the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 22.6.3.4, “Debug Store (DS) Mechanism.”		If (CPUID.01H:EDX.DS[21] = 1)
63:0	The linear address of the first byte of the DS buffer management area, if IA-32e mode is active.	
31:0	The linear address of the first byte of the DS buffer management area, if not in IA-32e mode.	
63:32	Reserved if not in IA-32e mode.	
Register Address: 6A0H, 1696		IA32_U_CET
Configure User Mode CET (R/W)		Bits 1:0 are defined if CPUID.07H.00H:ECX.CET_SS[7] = 1. Bits 5:2 and bits 63:10 are defined if CPUID.07H.00H:EDX.CET_IBT[20] = 1.
0	SH_STK_EN: When set to 1, enable shadow stacks at CPL3.	
1	WR_SHSTK_EN: When set to 1, enables the WRSSD/WRSSQ instructions.	
2	ENDBR_EN: When set to 1, enables indirect branch tracking.	
3	LEG_IW_EN: Enable legacy compatibility treatment for indirect branch tracking.	
4	NO_TRACK_EN: When set to 1, enables use of no-track prefix for indirect branch tracking.	
5	SUPPRESS_DIS: When set to 1, disables suppression of CET indirect branch tracking on legacy compatibility.	
9:6	Reserved; must be zero.	
10	SUPPRESS: When set to 1, indirect branch tracking is suppressed. This bit can be written to 1 only if TRACKER is written as IDLE.	
11	TRACKER: Value of the indirect branch tracking state machine. Values: IDLE (0), WAIT_FOR_ENDBRANCH(1).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:12	EB_LEG_BITMAP_BASE: Linear address bits 63:12 of a legacy code page bitmap used for legacy compatibility when indirect branch tracking is enabled. If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are used.	
Register Address: 6A2H, 1698		IA32_S_CET
Configure Supervisor Mode CET (R/W)		See IA32_U_CET (6A0H) for reference; similar format.
Register Address: 6A4H, 1700		IA32_PL0_SSP (also called IA32_FRED_SSPO)
Linear address to be loaded into SSP on transition to privilege level 0. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0. Transitions to privilege level 0 will check that bit 2 is also 0. FRED event delivery uses this MSR as IA32_FRED_SSPO: if FRED event delivery causes a transition from CPL 3, SSP is loaded from this MSR.		If CPUID.07H.00H:ECX.CET_SS[7] = 1
Register Address: 6A5H, 1701		IA32_PL1_SSP
Linear address to be loaded into SSP on transition to privilege level 1. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0. Transitions to privilege level 1 from a higher privilege level will check that bit 2 is also 0.		If CPUID.07H.00H:ECX.CET_SS[7] = 1
Register Address: 6A6H, 1702		IA32_PL2_SSP
Linear address to be loaded into SSP on transition to privilege level 2. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0. Transitions to privilege level 2 from a higher privilege level will check that bit 2 is also 0.		If CPUID.07H.00H:ECX.CET_SS[7] = 1
Register Address: 6A7H, 1703		IA32_PL3_SSP
Linear address to be loaded into SSP on transition to privilege level 3. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0.		If CPUID.07H.00H:ECX.CET_SS[7] = 1
Register Address: 6A8H, 1704		IA32_INTERRUPT_SSP_TABLE_ADDR
Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) This MSR is not present on processors that do not support Intel 64 architecture. This field cannot represent a non-canonical address.		If CPUID.07H.00H:ECX.CET_SS[7] = 1
Register Address: 6E0H, 1760		IA32_TSC_DEADLINE
TSC Target of Local APIC's TSC Deadline Mode (R/W)		If CPUID.01H:ECX[24] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:0	REGISTER_VALUE TSC-deadline value.	
Register Address: 6E1H, 1761		IA32_PKRS
Specifies the PK permissions associated with each protection domain for supervisor pages (R/W)		If CPUID.07H:00H:ECX.PKS[31] = 1
31:0	For domain i (i between 0 and 15), bits 2i and 2i+1 contain the AD and WD permissions, respectively.	
63:32	Reserved.	
Register Address: 770H, 1904		IA32_PM_ENABLE
Enable/disable HWP (R/W)		If CPUID.06H:EAX[7] = 1
0	HWP_ENABLE (R/W) Note this bit can only be enabled once from the default value. Once set, writes to the HWP_ENABLE bit are ignored. Only RESET will clear this bit. Default = 0. See Section 17.4.2, "Enabling HWP."	If CPUID.06H:EAX[7] = 1
63:1	Reserved.	
Register Address: 771H, 1905		IA32_HWP_CAPABILITIES
HWP Performance Range Enumeration (R/O)		If CPUID.06H:EAX[7] = 1
7:0	Highest_Performance See Section 17.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
15:8	Guaranteed_Performance See Section 17.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
23:16	Most_Efficient_Performance See Section 17.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
31:24	Lowest_Performance See Section 17.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
63:32	Reserved.	
Register Address: 772H, 1906		IA32_HWP_REQUEST_PKG
Power Management Control Hints for All Logical Processors in a Package (R/W)		If CPUID.06H:EAX[11] = 1
7:0	Minimum_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1
15:8	Maximum_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1
23:16	Desired_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1
31:24	Energy_Performance_Preference See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1 && CPUID.06H:EAX[10] = 1
41:32	Activity_Window See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1 && CPUID.06H:EAX[9] = 1
63:42	Reserved.	
Register Address: 773H, 1907		IA32_HWP_INTERRUPT
Control HWP Native Interrupts (R/W)		If CPUID.06H:EAX[8] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
0	EN_Guaranteed_Performance_Change See Section 17.4.6, "HWP Notifications."	If CPUID.06H:EAX[8] = 1
1	EN_Excursion_Minimum See Section 17.4.6, "HWP Notifications."	If CPUID.06H:EAX[8] = 1
63:2	Reserved.	
Register Address: 774H, 1908		IA32_HWP_REQUEST
Power Management Control Hints to a Logical Processor (R/W)		If CPUID.06H:EAX[7] = 1
7:0	Minimum_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1
15:8	Maximum_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1
23:16	Desired_Performance See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1
31:24	Energy_Performance_Preference See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1 && CPUID.06H:EAX[10] = 1
41:32	Activity_Window See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1 && CPUID.06H:EAX[9] = 1
42	Package_Control See Section 17.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1 && CPUID.06H:EAX[11] = 1
63:43	Reserved.	
Register Address: 775H, 1909		IA32_PECI_HWP_REQUEST_INFO
IA32_PECI_HWP_REQUEST_INFO		
7:0	Minimum Performance (MINIMUM_PERFORMANCE): Used by OS to read the latest value of Peci minimum performance input. Default value is 0.	
15:8	Maximum Performance (MAXIMUM_PERFORMANCE): Used by OS to read the latest value of Peci maximum performance input. Default value is 0.	
23:16	Reserved.	
31:24	Energy Performance Preference (ENERGY_PERFORMANCE_PREFERENCE): Used by OS to read the latest value of Peci Energy Performance Preference input. Default value is 0.	
59:32	Reserved.	
60	EPP Peci Override (EPP_PECI_OVERRIDE): Indicates whether Peci is currently overriding the Energy Performance Preference input. If set to '1', Peci is overriding the Energy Performance Preference input. If clear (0), OS has control over Energy Performance Preference input. Default value is 0.	
61	Reserved.	
62	Max Peci Override (MAX_PECI_OVERRIDE): Indicates whether Peci is currently overriding the Maximum Performance input. If set to '1', Peci is overriding the Maximum Performance input. If clear (0), OS has control over Maximum Performance input. Default value is 0.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63	Min PECL Override (MIN_PECI_OVERRIDE): Indicates whether PECL is currently overriding the Minimum Performance input. If set to '1', PECL is overriding the Minimum Performance input. If clear (0), OS has control over Minimum Performance input. Default value is 0.	
Register Address: 776H, 1910		IA32_HWP_CTL
IA32_HWP_CTL		If CPUID.06H:EAX[22] = 1
0	PKG_CTL_POLARITY Defines which HWP Request MSR is used whether Thread level or package level. When package MSR is used, the thread MSR valid bits define which thread MSR fields override the package. Default value is 0.	If CPUID.06H:EAX[22] = 1
63:1	Reserved.	
Register Address: 777H, 1911		IA32_HWP_STATUS
Log bits indicating changes to Guaranteed & excursions to Minimum (R/W)		If CPUID.06H:EAX[7] = 1
0	Guaranteed_Performance_Change (R/WCO) See Section 17.4.5, "HWP Feedback."	If CPUID.06H:EAX[7] = 1
1	Reserved.	
2	Excursion_To_Minimum (R/WCO) See Section 17.4.5, "HWP Feedback."	If CPUID.06H:EAX[7] = 1
63:3	Reserved.	
Register Address: 7A3H, 1955		IA32_MCU_EXT_SERVICE
MCU Extended Service (R/O)		If IA32_ARCH_CAPABILITIES[22] = 1
3:0	ALLOWED_PERIODS Value indicates the allowed periods for extended servicing. Value x means that all extended servicing periods are allowed till period x.	
63:4	Reserved.	
Register Address: 7A4H, 1956		IA32_MCU_ROLLBACK_MIN_ID
Minimal MCU Revision ID (R/O) Minimal MCU Revision ID that software can rollback to per boot.		If IA32_MCU_ENUMERATION[3] = 1
31:0	REVISION_ID Minimal MCU revision ID for rollback.	
63:32	Reserved for future use.	
Register Address: 7A5H, 1957		IA32_MCU_STAGING_MBOX_ADDR
IA32_MCU_STAGING_MBOX_ADDR (R/O) Reports MMIO address of MCU staging DOE mailbox.		
63:0	ADDR MMIO address base of MCU staging DOE mailbox.	
Register Address: 7B0H, 1968		IA32_ROLLBACK_SIGN_ID_0

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Rollback ID 0 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B1H, 1969		IA32_ROLLBACK_SIGN_ID_1
Rollback ID 1 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B2H, 1970		IA32_ROLLBACK_SIGN_ID_2
Rollback ID 2 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B3H, 1971		IA32_ROLLBACK_SIGN_ID_3
Rollback ID 3 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B4H, 1972		IA32_ROLLBACK_SIGN_ID_4
Rollback ID 4 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 7B5H, 1973		IA32_ROLLBACK_SIGN_ID_5
Rollback ID 5 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B6H, 1974		IA32_ROLLBACK_SIGN_ID_6
Rollback ID 6 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B7H, 1975		IA32_ROLLBACK_SIGN_ID_7
Rollback ID 7 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B8H, 1976		IA32_ROLLBACK_SIGN_ID_8
Rollback ID 8 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B9H, 1977		IA32_ROLLBACK_SIGN_ID_9
Rollback ID 9 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:48	Reserved.	
Register Address: 7BAH, 1978		IA32_ROLLBACK_SIGN_ID_10
Rollback ID 10 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BBH, 1979		IA32_ROLLBACK_SIGN_ID_11
Rollback ID 11 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BCH, 1980		IA32_ROLLBACK_SIGN_ID_12
Rollback ID 12 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BDH, 1981		IA32_ROLLBACK_SIGN_ID_13
Rollback ID 13 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BEH, 1982		IA32_ROLLBACK_SIGN_ID_14
Rollback ID 14 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BFH, 1983		IA32_ROLLBACK_SIGN_ID_15
Rollback ID 15 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 802H, 2050		IA32_X2APIC_APICID
x2APIC ID Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 803H, 2051		IA32_X2APIC_VERSION
x2APIC Version Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 808H, 2056		IA32_X2APIC_TPR
x2APIC Task Priority Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80AH, 2058		IA32_X2APIC_PPR
x2APIC Processor Priority Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80BH, 2059		IA32_X2APIC_EOI
x2APIC EOI Register (W/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80DH, 2061		IA32_X2APIC_LDR
x2APIC Logical Destination Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80FH, 2063		IA32_X2APIC_SIVR
x2APIC Spurious Interrupt Vector Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 810H, 2064		IA32_X2APIC_ISR0
x2APIC In-Service Register Bits 31:0 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 811H, 2065		IA32_X2APIC_ISR1
x2APIC In-Service Register Bits 63:32 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 812H, 2066		IA32_X2APIC_ISR2
x2APIC In-Service Register Bits 95:64 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service Register Bits 127:96 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service Register Bits 159:128 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service Register Bits 191:160 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service Register Bits 223:192 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service Register Bits 255:224 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits 31:0 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 819H, 2073	IA32_X2APIC_TMR1	
x2APIC Trigger Mode Register Bits 63:32 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode Register Bits 95:64 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode Register Bits 127:96 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode Register Bits 159:128 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode Register Bits 191:160 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode Register Bits 223:192 (R/O)	If (CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1)	
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode Register Bits 255:224 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 820H, 2080	IA32_X2APIC_IRRO	
x2APIC Interrupt Request Register Bits 31:0 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 821H, 2081	IA32_X2APIC_IRR1	
x2APIC Interrupt Request Register Bits 63:32 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request Register Bits 95:64 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request Register Bits 127:96 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request Register Bits 159:128 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request Register Bits 191:160 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request Register Bits 223:192 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request Register Bits 255:224 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 836H, 2102		IA32_X2APIC_LVT_LINT1
x2APIC LVT LINT1 Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 837H, 2103		IA32_X2APIC_LVT_ERROR
x2APIC LVT Error Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 838H, 2104		IA32_X2APIC_INIT_COUNT
x2APIC Initial Count Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 839H, 2105		IA32_X2APIC_CUR_COUNT
x2APIC Current Count Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 83EH, 2110		IA32_X2APIC_DIV_CONF
x2APIC Divide Configuration Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 83FH, 2111		IA32_X2APIC_SELF_IPI
x2APIC Self IPI Register (W/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 981H, 2433		IA32_TME_CAPABILITY
Memory Encryption Capability MSR		If CPUID.07H.00H:ECX[13] = 1
0	Support for AES-XTS 128-bit encryption algorithm. (NIST standard)	
1	Support for AES-XTS 128-bit encryption with integrity algorithm.	
2	Support for AES-XTS 256-bit encryption algorithm.	
3	Support for AES-XTS 256-bit encryption with integrity algorithm.	
27:4	Reserved.	
28	Non-zero KeyIDs must be SEAM-private.	If set, any write to IA32_TME_ACTIVATE must write identical values to MK_TME_KEYID_BITS and TDX_RESERVED_KEYID_BITS.
29	Reserved.	
30	SUPPORT_IA32_TME_CLEAR_SAVED_KEY Support for the IA32_TME_CLEAR_SAVED_KEY MSR.	
31	TME encryption bypass supported.	
35:32	MK_TME_MAX_KEYID_BITS Number of bits which can be allocated for usage as key identifiers for multi-key memory encryption. 4 bits allow for a maximum value of 15, which could address 32K keys. Zero if TME-MK is not supported.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
50:36	MK_TME_MAX_KEYS Indicates the maximum number of keys which are available for usage. This value may not be a power of 2. KeyID 0 is specially reserved and is not accounted for in this field.	
63:51	Reserved.	
Register Address: 982H, 2434		IA32_TME_ACTIVATE
Memory Encryption Activation MSR This MSR is used to lock the MSRs listed below. Any write to the following MSRs will be ignored after they are locked. The lock is reset when CPU is reset. ▪ IA32_TME_ACTIVATE ▪ IA32_TME_EXCLUDE_MASK ▪ IA32_TME_EXCLUDE_BASE Note that IA32_TME_EXCLUDE_MASK and IA32_TME_EXCLUDE_BASE must be configured before IA32_TME_ACTIVATE.		If CPUID.07H.00H:ECX[13] = 1
0	Lock R/O – Will be set upon successful WRMSR (or first SMI); written value ignored.	
1	Hardware Encryption Enable This bit also enables TME-MK; TME-MK cannot be enabled without enabling encryption hardware. TME is enabled depending on the TME Encryption Bypass Enable bit (bit 31).	
2	Key Select 0: Create a new TME key (expected cold/warm boot). 1: Restore the TME key from storage (expected when resume from standby).	
3	Save TME Key for Standby Save key into storage to be used when resume from standby. Note: This may not be supported in all processors.	
7:4	TME Policy/Encryption Algorithm Only algorithms enumerated in IA32_TME_CAPABILITY MSR are allowed. Any other values are invalid and will result in #GP. Additionally, any algorithm that supports integrity checking is not allowed to be used for TME even if it is listed as allowed in the IA32_TME_CAPABILITY MSR and will result in #GP.	
30:8	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
31	TME Encryption Bypass Enable When encryption hardware is enabled: - Total Memory Encryption is enabled using a CPU generated ephemeral key based on a hardware random number generator when this bit is set to 0. - Total Memory Encryption is bypassed (no encryption/decryption for KeyID0) when this bit is set to 1. On some processors, bypassing TME encryption can provide performance benefits to accesses made with KeyID 0 by avoiding the latency of decryption or encryption and decryption. Software must inspect Hardware Encryption Enable (bit 1) and TME Encryption Bypass Enable (bit 31) to determine if TME encryption is enabled.	
35:32	MK_TME_KEYID_BITS Reserved if TME-MK is not enumerated, otherwise: The number of key identifier bits to allocate to TME-MK usage. Similar to enumeration, this is an encoded value. Writing a value greater than MK_TME_MAX_KEYID_BITS will result in #GP. Writing a non-zero value to this field will #GP if bit 1 of EAX (Hardware Encryption Enable) is not also set to 1, as encryption hardware must be enabled to use TME-MK. Example: To support 255 keys, this field would be set to a value of 8.	
39:36	TDX_RESERVED_KEYID_BITS The number of key identifier bits to allocate to TDX usage, which are allocated from the most significant bit downward. Writing a value greater than MK_TME_KEYID_BITS will result in a #GP. Note: These bits are a subset of the overall KeyID bits which are declared by MK_TME_MAX_KEYID_BITS.	If IA32_TME_CAPABILITY[28] = 1, this value must be the same as that of MK_TME_KEYID_BITS.
47:40	Reserved.	
63:48	MK_TME_CRYPT0_ALGS Reserved if TME-MK is not enumerated, otherwise: Bit 48: AES-XTS 128. Bit 49: AES-XTS 128 with integrity. Bit 50: AES-XTS 256. Bit 51: AES-XTS-256 with integrity. Bit 63:52: Reserved (#GP) Bitmask for BIOS to set which encryption algorithms are allowed for TME-MK, would be later enforced by the key loading ISA ('1' = allowed).	
Register Address: 983H, 2435		IA32_TME_EXCLUDE_MASK
Memory Encryption Exclude Mask The IA32_TME_EXCLUDE_MASK MSR must define a contiguous region. WRMSR will #GP if the TMEEMASK field does not specify a contiguous region. This MSR is locked by the IA32_TME_ACTIVATE MSR. If lock=1, then WRMSR to IA32_TME_EXCLUDE_MASK/IA32_TME_EXCLUDE_BASE MSRs will result in #GP.		If CPUID.07H.00H:ECX[13] = 1
10:0	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
11	Enable: When set to '1', then TME_EXCLUDE_BASE and TME_EXCLUDE_MASK are used to define an exclusion region for TME/TME-MK (for KeyID=0).	
MAXPHYADDR-1:12	TMEEMASK: This field indicates the bits that must match TMEEBASE in order to qualify as a TME/TME-MK (for KeyID=0) exclusion memory range access.	
63:MAXPHYADDR	Reserved; must be zero.	
Register Address: 984H, 2436		IA32_TME_EXCLUDE_BASE
Memory Encryption Exclude Base Note: Writing '1' into bits greater than the max supported physical size will result in #GP. This MSRs is locked by the IA32_TME_ACTIVATE MSR. If lock=1, then WRMSR to IA32_TME_EXCLUDE_MASK/IA32_TME_EXCLUDE_BASE MSRs will result in #GP.		If CPUID.07H.00H:ECX[13] = 1
11:0	Reserved.	
MAXPHYADDR-1:12	TMEEBASE: Base physical address to be excluded for TME/TME-MK (for KeyID=0) encryption.	
63:MAXPHYADDR	Reserved; must be zero.	
Register Address: 985H, 2437		IA32_UINTR_RR
User Interrupt Request Register (R/W)		If CPUID.07H.01H:EDX[13] = 1
63:0	UIRR Bitmap of requested user interrupt vectors.	
Register Address: 986H, 2438		IA32_UINTR_HANDLER
User Interrupt Handler Address (R/W)		If CPUID.07H.01H:EDX[13] = 1
63:0	UIHANDLER User interrupt handler linear address.	
Register Address: 987H, 2439		IA32_UINTR_STACKADJUST
User Interrupt Stack Adjustment (R/W)		If CPUID.07H.01H:EDX[13] = 1
0	LOAD_RSP User interrupt stack mode.	
2:1	Reserved.	
63:3	STACK_ADJUST Stack adjust value.	
Register Address: 988H, 2440		IA32_UINTR_MISC
User-Interrupt Target-Table Size and Notification Vector (R/W)		If CPUID.07H.01H:EDX[13] = 1
31:0	UITTSZ The highest index of a valid entry in the user-interrupt target table. Valid entries are indices 0..UITTSZ (inclusive).	
39:32	UINV User-interrupt notification vector.	
63:40	Reserved.	
Register Address: 989H, 2441		IA32_UINTR_PD
User Interrupt PID Address (R/W)		If CPUID.07H.01H:EDX[13] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
5:0	Reserved.	
63:6	UPIDADDR User-interrupt notification processing accesses a UPID at this linear address.	
Register Address: 98AH, 2442		IA32_UINTR_TT
User-Interrupt Target Table (R/W)		If CPUID.07H.01H:EDX[13] = 1
0	SENDUIPI_ENABLE User-interrupt target table is valid.	
3:1	Reserved.	
63:4	UITTADDR User-interrupt target table base linear address.	
Register Address: 990H, 2448		IA32_COPY_STATUS ⁴
Status of Most Recent Platform to Local or Local to Platform Copies (R/O)		If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	IWKEY_COPY_SUCCESSFUL Status of most recent copy to or from IwKeyBackup.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:1	Reserved.	
Register Address: 991H, 2449		IA32_IWKEYBACKUP_STATUS ⁵
Information about IwKeyBackup Register (R/O)		If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	Backup/Restore Valid Cleared when a write to IwKeyBackup is initiated, and then set when the latest write of IwKeyBackup has been written to storage that persists across S3/S4 sleep state. If S3/S4 is entered between when an IwKeyBackup write occurs and when this bit is set, then IwKeyBackup may not be recovered after S3/S4 exit. During S3/S4 sleep state exit (system wake up), this bit is cleared. It is set again when IwKeyBackup is restored from persistent storage and thus available to be copied to IwKey using IA32_COPY_PLATFORM_TO_LOCAL MSR. Another write to IwKeyBackup (via IA32_COPY_LOCAL_TO_PLATFORM MSR) may fail if a previous write has not yet set this bit.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
1	Reserved.	
2	Backup Key Storage Read/Write Error Updated prior to backup/restore valid being set. Set when an error is encountered while backing up or restoring a key to persistent storage.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
3	IwKeyBackup Consumed Set after the previous backup operation has been consumed by the platform. This does not indicate that the system is ready for a second IwKeyBackup write as the previous IwKeyBackup write may still need to set Backup/restore valid.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:4	Reserved.	
Register Address: 9F1H, 2545		IA32_TSE_CAPABILITY
IA32_TSE_CAPABILITY		If CPUID.07H.01H:EBX.PBNDKB[1] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
15:0	Supported encryption algorithms.	
23:16	TSE engine key sources supported.	
35:24	Reserved.	
50:36	TSE_MAX_KEYS. Indicates the maximum number of keys available.	
63:51	Reserved.	
Register Address: 9FBH, 2555		IA32_TME_CLEAR_SAVED_KEY
IA32_TME_CLEAR_SAVED_KEY (W/O)		
0	TME_CLEAR_SAVED_KEY Clear saved TME keys.	
63:1	Reserved.	
Register Address: C80H, 3200		IA32_DEBUG_INTERFACE
Silicon Debug Feature Control (R/W)		If CPUID.01H:ECX[11] = 1
0	Enable (R/W) BIOS set 1 to enable Silicon debug features. Default is 0.	If CPUID.01H:ECX[11] = 1
29:1	Reserved.	
30	Lock (R/W): If 1, locks any further change to the MSR. The lock bit is set automatically on the first SMI assertion even if not explicitly set by BIOS. Default is 0.	If CPUID.01H:ECX[11] = 1
31	Debug Occurred (R/O): This “sticky bit” is set by hardware to indicate the status of bit 0. Default is 0.	If CPUID.01H:ECX[11] = 1
63:32	Reserved.	
Register Address: C81H, 3201		IA32_L3_QOS_CFG
L3 QOS Configuration (R/W)		If (CPUID.10H.01H:ECX[2] = 1)
0	Enable (R/W) Set 1 to enable L3 CAT masks and CLOS to operate in Code and Data Prioritization (CDP) mode.	
63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
Register Address: C82H, 3202		IA32_L2_QOS_CFG
L2 QOS Configuration (R/W)		If (CPUID.10H.02H:ECX[2] = 1)
0	Enable (R/W) Set 1 to enable L2 CAT masks and CLOS to operate in Code and Data Prioritization (CDP) mode.	
63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
Register Address: C83H, 3203		IA32_L3_IO_QOS_CFG
L3 I/O QOS Configuration (R/W) This MSR is used to enable the I/O RDT features.		If (CPUID.0FH.01H:EAX[10:9] = 1)
0	L3 I/O RDT Allocation Enable.	
1	L3 I/O RDT Monitoring Enable.	
63:2	Reserved.	
Register Address: C88H, 3208		IA32_RESOURCE_PRIORITY

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Thread scope Resource Priority Enable (R/W)			
0	ENABLE When set, enables model specific features that can be used to create a Resource Priority mode.		
63:1	Reserved.		
Register Address: C89H, 3209		IA32_RESOURCE_PRIORITY_PKG	
IA32_RESOURCE_PRIORITY_PKG (R/W)			
0	ENABLE Enable Resource Priority feature.		
63:1	Reserved.		
Register Address: C8DH, 3213		IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W)			If (CPUID.07H.00H:EBX[12] = 1)
7:0	Event ID: ID of a supported monitoring event to report via IA32_QM_CTR.		
31: 8	Reserved.		
N+31:32	Resource Monitoring ID: ID for monitoring hardware to report monitored data via IA32_QM_CTR.		N = Ceil (Log ₂ (CPUID.0FH.00H:EBX[31:0] + 1))
63:N+32	Reserved.		
Register Address: C8EH, 3214		IA32_QM_CTR	
Monitoring Counter Register (R/O)			If (CPUID.07H.00H:EBX[12] = 1)
61:0	Resource Monitored Data.		
62	Unavailable: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.		
63	Error: If 1, indicates an unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.		
Register Address: C8FH, 3215		IA32_PQR_ASSOC	
Resource Association Register (R/W)			If ((CPUID.07H.00H:EBX[12] = 1) or (CPUID.07H.00H:EBX[15] = 1))
N-1:0	Resource Monitoring ID (R/W): ID for monitoring hardware to track internal operation, e.g., memory access.		N = Ceil (Log ₂ (CPUID.0FH.00H:EBX[31:0] + 1))
31:N	Reserved.		
63:32	CLOS (R/W): The class of service (CLOS) to enforce (on writes); returns the current CLOS when read.		If (CPUID.07H.00H:EBX[15] = 1)
Register Address: C90H—D8FH, 3216—3471		Reserved MSR Address Space for CAT Mask Registers	
See Section 20.19.4.1, “Enumeration and Detection Support of Cache Allocation Technology.”			
Register Address: C90H, 3216		IA32_L3_MASK_0	
L3 CAT Mask for COS0 (R/W)			If (CPUID.10H.00H:EBX[1] != 0)
31:0	Capacity Bit Mask (R/W)		
63:32	Reserved.		
Register Address: C90H+n, 3216+n		IA32_L3_MASK_n	
L3 CAT Mask for COSn (R/W)			n = CPUID.10H.01H:EDX[15:0]

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
31:0	Capacity Bit Mask (R/W)	
63:32	Reserved.	
Register Address: D10H–D4FH, 3344–3407		Reserved MSR Address Space for L2 CAT Mask Registers
See Section 20.19.4.1, “Enumeration and Detection Support of Cache Allocation Technology.”		
Register Address: D10H, 3344		IA32_L2_MASK_0
L2 CAT Mask for COS0 (R/W)		If (CPUID.10H.00H:EBX[2] != 0)
31:0	Capacity Bit Mask (R/W)	
63:32	Reserved.	
Register Address: D10H+n, 3344+n		IA32_L2_MASK_n
L2 CAT Mask for COSn (R/W)		n = CPUID.10H.02H:EDX[15:0]
31:0	Capacity Bit Mask (R/W)	
63:32	Reserved.	
Register Address: D18H, 3352		IA32_L2_MASK_8
L2 CAT Mask for COS8 (R/W)		
15:0	WAY_MASK Capacity Bit Mask. Available ways vectors for class of service of IA core. '1' in bit indicates allocation to the way is allowed. '0' indicates allocation to the way is not allowed.	
63:16	Reserved.	
Register Address: D19H, 3353		IA32_L2_MASK_9
L2 CAT Mask for COS9 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1AH, 3354		IA32_L2_MASK_10
L2 CAT Mask for COS10 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1BH, 3355		IA32_L2_MASK_11
L2 CAT Mask for COS11 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1CH, 3356		IA32_L2_MASK_12
L2 CAT Mask for COS12 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1DH, 3357		IA32_L2_MASK_13
L2 CAT Mask for COS13 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1EH, 3358		IA32_L2_MASK_14
L2 CAT Mask for COS14 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1FH, 3359		IA32_L2_MASK_15

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
L2 CAT Mask for COS15 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D50H, 3408		IA32_L2_QOS_EXT_BW_THRTL_0
IA32_L2_QOS_EXT_BW_THRTL_0 (R/W) Memory Bandwidth enforcement for COS0.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 0
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D51H, 3409		IA32_L2_QOS_EXT_BW_THRTL_1
IA32_L2_QOS_EXT_BW_THRTL_1 (R/W) Memory Bandwidth enforcement for COS1.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 1
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D52H, 3410		IA32_L2_QOS_EXT_BW_THRTL_2
IA32_L2_QOS_EXT_BW_THRTL_2 (R/W) Memory Bandwidth enforcement for COS2.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 2
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D53H, 3411		IA32_L2_QOS_EXT_BW_THRTL_3
IA32_L2_QOS_EXT_BW_THRTL_3 (R/W) Memory Bandwidth enforcement for COS3.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 3
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D54H, 3412		IA32_L2_QOS_EXT_BW_THRTL_4
IA32_L2_QOS_EXT_BW_THRTL_4 (R/W) Memory Bandwidth enforcement for COS4.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 4
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D55H, 3413		IA32_L2_QOS_EXT_BW_THRTL_5
IA32_L2_QOS_EXT_BW_THRTL_5 (R/W) Memory Bandwidth enforcement for COS5.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 5
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D56H, 3414		IA32_L2_QOS_EXT_BW_THRTL_6

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
IA32_L2_QOS_EXT_BW_THRTL_6 (R/W) Memory Bandwidth enforcement for COS6.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 6
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D57H, 3415		IA32_L2_QOS_EXT_BW_THRTL_7
IA32_L2_QOS_EXT_BW_THRTL_7 (R/W) Memory Bandwidth enforcement for COS7.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 7
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D58H, 3416		IA32_L2_QOS_EXT_BW_THRTL_8
IA32_L2_QOS_EXT_BW_THRTL_8 (R/W) Memory Bandwidth enforcement for COS8.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 8
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D59H, 3417		IA32_L2_QOS_EXT_BW_THRTL_9
IA32_L2_QOS_EXT_BW_THRTL_9 (R/W) Memory Bandwidth enforcement for COS9.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 9
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5AH, 3418		IA32_L2_QOS_EXT_BW_THRTL_10
IA32_L2_QOS_EXT_BW_THRTL_10 (R/W) Memory Bandwidth enforcement for COS10.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 10
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5BH, 3419		IA32_L2_QOS_EXT_BW_THRTL_11
IA32_L2_QOS_EXT_BW_THRTL_11 (R/W) Memory Bandwidth enforcement for COS11.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 11
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5CH, 3420		IA32_L2_QOS_EXT_BW_THRTL_12
IA32_L2_QOS_EXT_BW_THRTL_12 (R/W) Memory Bandwidth enforcement for COS12.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 12

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5DH, 3421		IA32_L2_QOS_EXT_BW_THRTL_13
IA32_L2_QOS_EXT_BW_THRTL_13 (R/W) Memory Bandwidth enforcement for COS13.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 13
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5EH, 3422		IA32_L2_QOS_EXT_BW_THRTL_14
IA32_L2_QOS_EXT_BW_THRTL_14 (R/W) Memory Bandwidth enforcement for COS14.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 14
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D90H, 3472		IA32_BNDCFGS
Supervisor State of MPX Configuration (R/W)		If (CPUID.07H.00H:EBX[14] = 1)
0	EN: Enable Intel MPX in supervisor mode.	
1	BNDPRESERVE: Preserve the bounds registers for near branch instructions in the absence of the BND prefix.	
11:2	Reserved, must be zero.	
63:12	Base Address of Bound Directory.	
Register Address: D91H, 3473		IA32_COPY_LOCAL_TO_PLATFORM ⁵
Copy Local State to Platform State (W)		If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	lWKeyBackup Copy lWKey to lWKeyBackup.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:1	Reserved.	
Register Address: D92H, 3474		IA32_COPY_PLATFORM_TO_LOCAL ⁵
Copy Platform State to Local State (W)		If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	lWKeyBackup Copy lWKeyBackup to lWKey.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:1	Reserved.	
Register Address: D93H, 3475		IA32_PASID
Process Address Space Identifier. (R/W)		
19:0	Process address space identifier (PASID). Specifies the PASID of the currently running software thread.	
30:20	Reserved.	
31	Valid. Execution of ENQCMD causes a #GP if this bit is clear.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:32	Reserved.	
Register Address: DA0H, 3488		IA32_XSS
Extended Supervisor State Mask (R/W)		If(CPUID.0DH:01H:EAX[3] = 1
7:0	Reserved.	
8	PT State (R/W)	
9	Reserved.	
10	PASID State (R/W)	
11	CET_U State (R/W)	
12	CET_S State (R/W)	
13	HDC State (R/W)	
14	UINTR State (R/W)	
15	LBR State (R/W)	
16	HWP State (R/W)	
63:17	Reserved.	
Register Address: DB0H, 3504		IA32_PKG_HDC_CTL
Package Level Enable/Disable HDC (R/W)		If CPUID.06H:EAX[13] = 1
0	HDC_Pkg_Enable (R/W) Force HDC idling or wake up HDC-idled logical processors in the package. See Section 17.5.2, "Package level Enabling HDC."	If CPUID.06H:EAX[13] = 1
63:1	Reserved.	
Register Address: DB1H, 3505		IA32_PM_CTL1
Enable/Disable the HDC Thread Level Activity (R/W)		If CPUID.06H:EAX[13] = 1
0	SDC_ALLOWED (R/W) Set this bit to allow this thread to be forced into HDC idle state. Clearing this bit blocks HDC-enter (HW) request. Default value: 1. See Section 17.5.3.	If CPUID.06H:EAX[13] = 1
63:1	Reserved.	
Register Address: DB2H, 3506		IA32_THREAD_STALL
Per-Logical_Processor_ID HDC Idle Residency (R/O)		If CPUID.06H:EAX[13] = 1
63:0	Stall_Cycle_Cnt (R/W) Stalled cycles due to HDC forced idle on this logical processor. See Section 17.5.4.1.	If CPUID.06H:EAX[13] = 1
Register Address: E00H, 3584		IA32_QOS_CORE_BW_THRTL_0
CBA Levels Based on COS for Bandwidth Throttling (R/W)		CPUID.10H:00H:EBX[5] = 1
3:0	COS0_LEVEL CBA Level for COS[0]. Levels are programmed from 0 to 15.	
7:4	Reserved.	
11:8	COS1_LEVEL CBA Level for COS[1]. Levels are programmed from 0 to 15.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
15:12	Reserved.	
19:16	COS2_LEVEL CBA Level for COS[2]. Levels are programmed from 0 to 15.	
25:20	Reserved.	
27:24	COS3_LEVEL CBA Level for COS[3]. Levels are programmed from 0 to 15.	
31:28	Reserved.	
35:32	COS4_LEVEL CBA Level for COS[4]. Levels are programmed from 0 to 15.	
39:36	Reserved.	
43:40	COS5_LEVEL CBA Level for COS[5]. Levels are programmed from 0 to 15.	
47:44	Reserved.	
51:48	COS6_LEVEL CBA Level for COS[6]. Levels are programmed from 0 to 15.	
Register Address: E01H, 3585		IA32_QOS_CORE_BW_THRTL_1
CBA Levels Based on COS for Bandwidth Throttling (R/W)		CPUID.10H.00H:EBX[5] = 1
3:0	COS8_LEVEL CBA Level for COS[8]. Levels are programmed from 0 to 15.	
7:4	Reserved.	
11:8	COS9_LEVEL CBA Level for COS[9]. Levels are programmed from 0 to 15.	
15:12	Reserved.	
19:16	COS10_LEVEL CBA Level for COS[10]. Levels are programmed from 0 to 15.	
25:20	Reserved.	
27:24	COS11_LEVEL CBA Level for COS[11]. Levels are programmed from 0 to 15.	
31:28	Reserved.	
35:32	COS12_LEVEL CBA Level for COS[12]. Levels are programmed from 0 to 15.	
39:36	Reserved.	
43:40	COS13_LEVEL CBA Level for COS[13]. Levels are programmed from 0 to 15.	
47:44	Reserved.	
51:48	COS14_LEVEL CBA Level for COS[14]. Levels are programmed from 0 to 15.	
55:50	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
59:56	COS15_LEVEL CBA Level for COS[15]. Levels are programmed from 0 to 15.	
63:60	Reserved	
Register Address: 1200H–121FH, 4608–4639		IA32_LBR_x_INFO
Last Branch Record Entry X Info Register (R/W) An attempt to read or write IA32_LBR_x_INFO such that $x \geq \text{IA32\_LBR\_DEPTH.DEPTH}$ will #GP.		
15:0	CYC_CNT The elapsed CPU cycles (saturating) since the last LBR was recorded. See Section 18.1.3.3.	Reset Value: 0
55:16	Undefined, may be zero or non-zero. Writes of non-zero values do not fault, but reads may return a different value.	Reset Value: 0
59:56	BR_TYPE The branch type recorded by this LBR. Encodings: 0000B: COND 0001B: JMP Indirect 0010B: JMP Direct 0011B: CALL Indirect 0100B: CALL Direct 0101B: RET 011xB: Reserved 1xxxB: Other Branch	Reset Value: 0
60	CYC_CNT_VALID CYC_CNT value is valid. See Section 21.1.3.3.	Reset Value: 0
61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
62	IN_TSX This LBR record records a branch that retired during a TSX transaction. On processors that do not support Intel TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
63	MISPRED The recorded branch direction (conditional branch) or target (indirect branch) was mispredicted.	Reset Value: 0
Register Address: 1400H, 5120		IA32_SEAMRR_BASE
SEAM Memory Range Register for TDX - Base Address (R/W)		IA32_MTRRCAP.SEAMRR[15] = 1
2:0	Reserved.	
3	CONFIGURED When set to 1, the SEAM range is configured.	
24:4	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
M-1:25	BASE SEAM Range Register BASE address.	M is the value enumerated by CPUID.80000008H:EAX[7:0] Bits M-1:M-k must be 0, where k = IA32_TME_ACTIVATE.MK_TME_KEYID_BITS
63:M	Reserved.	
Register Address: 1401H, 5121		IA32_SEAMRR_MASK
SEAM Memory Range Register - Address Mask (R/W)		IA32_MTRRCAP.SEAMRR[15] = 1
9:0	Reserved.	
10	LOCK	When the LOCK bit is set, the IA32_SEAMRR_BASE/MASK registers are non-writable.
11	VALID Returns 1 when SEAM range protections are active	Read-only. Must be written with 0.
24:12	Reserved.	
M-1:25	MASK Mask value for SEAMRR matching. Lowest granularity is 32MB.	M is the value enumerated by CPUID.80000008H:EAX[7:0]
63:M	Reserved.	
Register Address: 1406H, 5126		IA32_MCU_CONTROL
MCU Control (R/W) Controls the behavior of the Microcode Update Trigger MSR, IA32_BIOS_UPDT_TRIG.		If CPUID.07H.00H:EDX[29] = 1 && IA32_ARCH_CAPABILITIES.MCU_CONTROL = 1
0	LOCK Once set, further writes to this MSR will cause a #GP(0) fault. Bypassed during SMM if EN_SMM_BYPASS (bit 2) is set.	
1	DIS_MCU_LOAD If this bit is set on a given logical processor, then any subsequent attempts to load a microcode update by that logical processor will be silently dropped (WRMSR 0x79 has no effect).	
2	EN_SMM_BYPASS If set, then writes to IA32_MCU_CONTROL are allowed during SMM regardless of the LOCK bit. This enables BIOS to Opt-In to the SMM Bypass functionality.	
63:3	Reserved.	
Register Address: 14CEH, 5326		IA32_LBR_CTL
Last Branch Record Enabling and Configuration Register (R/W)		
0	LBREn When set, enables LBR recording.	Reset Value: 0
1	OS When set, allows LBR recording when CPL == 0.	Reset Value: 0
2	USR When set, allows LBR recording when CPL != 0.	Reset Value: 0

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
3	CALL_STACK When set, records branches in call-stack mode. See Section 21.1.2.4.	Reset Value: 0
15:4	Reserved.	Reset Value: 0
16	COND When set, records taken conditional branches. See Section 21.1.2.3.	
17	NEAR_REL_JMP When set, records near relative JMPs. See Section 21.1.2.3.	
18	NEAR_IND_JMP When set, records near indirect JMPs. See Section 21.1.2.3.	
19	NEAR_REL_CALL When set, records near relative CALLs. See Section 21.1.2.3.	
20	NEAR_IND_CALL When set, records near indirect CALLs. See Section 21.1.2.3.	
21	NEAR_RET When set, records near RETs. See Section 21.1.2.3.	
22	OTHER_BRANCH When set, records other branches. See Section 21.1.2.3.	
63:23	Reserved.	
Register Address: 14CFH, 5327		IA32_LBR_DEPTH
Last Branch Record Maximum Stack Depth Register (R/W)		
N:0	DEPTH The number of LBRs to be used for recording. Supported values are indicated by the bitmap in CPUID.1CH.00H:EAX[7:0]. The reset value will match the maximum supported by the CPU. Writes of unsupported values will #GP fault.	Reset Value: Varies
63:N+1	Reserved.	Reset Value: 0
Register Address: 1500H–151FH, 5376–5407		IA32_LBR_x_FROM_IP
Last Branch Record entry X source IP register (R/W). An attempt to read or write IA32_LBR_x_FROM_IP such that $x \geq \text{IA32\_LBR\_DEPTH.DEPTH}$ will #GP.		
63:0	FROM_IP The source IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	Reset Value: 0
Register Address: 1600H–161FH, 5632–5663		IA32_LBR_x_TO_IP
Last Branch Record Entry X Destination IP Register (R/W) An attempt to read or write IA32_LBR_x_TO_IP such that $x \geq \text{IA32\_LBR\_DEPTH.DEPTH}$ will #GP.		
63:0	TO_IP The destination IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	Reset Value: 0
Register Address: 17D0H, 6096		IA32_HW_FEEDBACK_PTR
Hardware Feedback Interface Pointer		If CPUID.06H:EAX[19] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
0	Valid (R/W) When set to 1, indicates a valid pointer is programmed into the ADDR field of the MSR.	
11:1	Reserved.	
MAXPHYADDR-1:12	ADDR (R/W) Physical address of the page frame of the first page of the hardware feedback interface structure.	
63:MAXPHYADDR	Reserved.	
Register Address: 17D1H, 6097		IA32_HW_FEEDBACK_CONFIG
Hardware Feedback Interface Configuration		If CPUID.06H:EAX[19] = 1
0	Enable (R/W) When set to 1, enables the hardware feedback interface.	
63:1	Reserved.	
Register Address: 17D2H, 6098		IA32_THREAD_FEEDBACK_CHAR
Thread Feedback Characteristics (R/O)		If CPUID.06H:EAX[23] = 1
7:0	Application Class ID, pointing into the Intel Thread Director structure.	
62:8	Reserved.	
63	Valid bit. When set to 1 the OS Scheduler can use the Class ID (in bits 7:0) for its scheduling decisions. If this bit is 0, the Class ID field should be ignored. It is recommended that the OS uses the last known Class ID of the software thread for its scheduling decisions.	
Register Address: 17D4H, 6100		IA32_HW_FEEDBACK_THREAD_CONFIG
Hardware Feedback Thread Configuration (R/W)		
0	Enables Intel Thread Director. When set to 1, logical processor scope Intel Thread Director is enabled. Default is 0 (disabled).	
63:1	Reserved.	
Register Address: 17DAH, 6106		IA32_HRESET_ENABLE
History Reset Enable (R/W)		
0	Enable reset of the Intel Thread Director history.	
31:1	Reserved for other capabilities that can be reset by the HRESET instruction.	
63:32	Reserved.	
Register Address: 1900H, 6400		IA32_PMC_GPO_CTR
Full Width Writable General Performance Counter 0 (R/W)		If CPUID.0AH:EAX[15:8] > 0 and IA32_PERF_CAPABILITIES[13] = 1
47:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:48	Reserved.	
Register Address: 1901H, 6401		IA32_PMC_GPO_CFG_A

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
IA32_PMC_GPO_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 0.		If CPUID.0AH:EAX[15:8] > 0
7:0	EVENT_SELECT Selects a performance event logic unit.	
15:8	UMASK Qualifies the microarchitectural condition to detect on the selected event logic.	
16	USR When set, events are counted only when the processor is operating at privilege levels 1, 2 or 3. This flag can be used in conjunction with the OS flag.	
17	OS When set, events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USER flag.	
18	EDGE When set, enables edge detection of events.	
19	Reserved.	
20	INT When set, the processor generates an exception through its local APIC on counter overflow for this counter's thread.	
21	ANYTHREAD If CPUID.0AH:EDX[15] is 1, then this bit is deprecated. When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	
22	ENABLE When set, performance counting is enabled in the performance-monitoring counter; when clear, the counter is disabled.	
23	INVERT Inverts the result of the counter-mask (CMASK) comparison when set, so that both greater than equal to and less than comparisons can be made. 0: The comparison is: threshold is greater than or equal to the event 1: The comparison is inverted: threshold is less than event.	
31:24	CMASK When CMASK is not zero, the corresponding performance counter increments by 1 each cycle if the event count is >= CMASK. This mask enables counting cycles in which multiple occurrences happen (for example, two or more instructions retired per clock).	
34:32	Reserved.	
35	EN_LBR_LOG When set enables updating LBRs with that counters event occurrences, if selected event is precise.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
36	EQUAL When EQ flag is set and the INV flag is clear, the comparison evaluates to true if the selected performance monitoring event (the event) is equal to the specified Counter Mask value (CMask). When EQ flag is set and INV flag is set, the comparison evaluates to true if the event is less-than the CMask value and the event is not zero. Note if CMask is zero, the EQ flag is ignored.	
39:37	Reserved.	
47:40	UMASK2 Unit mask 2 (UMASK2) field (bits 40 through 47) - These bits qualify the condition that the selected event logic unit detects. Valid UMASK2 values for each event logic unit are specific to the unit. The new UMASK2 field may also be used in conjunction with UMASK.	
63:48	Reserved.	
Register Address: 1903H, 6403		IA32_PMC_GPO_CFG_C
IA32_PMC_GPO_CFG_C (R/W) Extended Perf event selector for GP counter 0.		CPUID.23H.05H:EAX[0] = 1 or CPUID.23H.02H:EAX[0] = 1
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
34:32	Reserved.	
35	ALLOW_IN_RECORD (R/W) If 1, indicates that this counter's value will be included in the PEBS record if the associated CNTR subgroup is enabled.	
38:36	CNTR (R/W) When any bit is set, the counter group will be included in the PEBS record. - Bit 36: If 1, the general-purpose counters subgroup will be included. - Bit 37: If 1, the fixed-function counters subgroup will be included. - Bit 38: If 1, the performance metrics subgroup will be included and will be reset	
39	Reserved.	
41:40	LBR (R/W) Identifies the number of Last Branch Records (LBRs) to record. The encodings of this field are as follows: 00 - LBR group will not be recorded. The group is disabled. 01 - 8 LBRs will be recorded. 10 - 16 LBRs will be recorded. 11 - The number of LBRs to be recorded is IA32_LBR_DEPTH.DEPTH.	
48:42	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
55:49	XER (R/W) When set, enables particular XSAVE-Enabled Registers to be included in the PEBS record. The encodings for the register subgroup are: ▪ Bit 49 – SSER: Record XMM0-XMM15. ▪ Bit 50 – YMMHIR: Record the upper 128 bits of YMM0-YMM15. ▪ Bit 51 – EGPR: Record R16-R31. ▪ Bit 52 – Reserved and must be zero. ▪ Bit 53 – OPMASKR: Record K0-K7. ▪ Bit 54 – ZMMHIR: Record the upper 256 bits of ZMM0-ZMM15. ▪ Bit 55 – Hi16ZMMR: Record ZMM16-ZMM31.	
60:56	Reserved.	
61	GPR (R/W) If 1, enables the General-Purpose Registers Group to be included in the PEBS record.	
62	AUX (R/W) If 1, enables the Auxiliary Group to be included in the PEBS record.	
63	PEBS (R/W) If 1, PEBS is enabled for this counter.	
Register Address: 1904H, 6404		IA32_PMC_GP1_CTR
Full Width Writable General Performance Counter 1 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 1 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1905H, 6405		IA32_PMC_GP1_CFG_A
IA32_PMC_GP1_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 1. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 1
Register Address: 1907H, 6407		IA32_PMC_GP1_CFG_C
IA32_PMC_GP1_CFG_C (R/W) Extended Perf event selector for GP counter 1. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[1] = 1 or CPUID.23H.02H:EAX[1] = 1
Register Address: 1908H, 6408		IA32_PMC_GP2_CTR
Full Width Writable General Performance Counter 2 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 2 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1909H, 6409		IA32_PMC_GP2_CFG_A
IA32_PMC_GP2_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 2. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 2
Register Address: 190AH, 6410		IA32_PMC_GP2_CFG_B
IA32_PMC_GP2_CFG_B (R/W) GP counter reload configuration register.		
1:0	Reserved.	
2	RELOAD_PMC2 Reload GP2 when GP2 overflows.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
3	RELOAD_PMC3 Reload GP2 when GP3 overflows.	
4	RELOAD_PMC4 Reload GP2 when GP4 overflows.	
5	RELOAD_PMC5 Reload GP2 when GP5 overflows.	
6	RELOAD_PMC6 Reload GP2 when GP6 overflows.	
7	RELOAD_PMC7 Reload GP2 when GP7 overflows.	
31:8	Reserved.	
32	RELOAD_FC0 Reload GP2 when FC0 overflows.	
33	RELOAD_FC1 Reload GP2 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of GP2.	
63:49	Reserved.	
Register Address: 190BH, 6411		IA32_PMC_GP2_CFG_C
IA32_PMC_GP2_CFG_C (R/W) Extended Perf event selector for GP counter 2. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[2] = 1 or CPUID.23H.02H:EAX[2] = 1
Register Address: 190CH, 6412		IA32_PMC_GP3_CTR
Full Width Writable General Performance Counter 3 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 3 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 190DH, 6413		IA32_PMC_GP3_CFG_A
IA32_PMC_GP3_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 3. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 3
Register Address: 190EH, 6414		IA32_PMC_GP3_CFG_B
IA32_PMC_GP3_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 190FH, 6415		IA32_PMC_GP3_CFG_C
IA32_PMC_GP3_CFG_C (R/W) Extended Perf event selector for GP counter 3. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[3] = 1 or CPUID.23H.02H:EAX[3] = 1
Register Address: 1910H, 6416		IA32_PMC_GP4_CTR

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Full Width Writable General Performance Counter 4 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 4 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1911H, 6417		IA32_PMC_GP4_CFG_A
IA32_PMC_GP4_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 4. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 4
Register Address: 1912H, 6418		IA32_PMC_GP4_CFG_B
IA32_PMC_GP4_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 1913H, 6419		IA32_PMC_GP4_CFG_C
IA32_PMC_GP4_CFG_C (R/W) Extended Perf event selector for GP counter 4. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[4] = 1 or CPUID.23H.02H:EAX[4] = 1
Register Address: 1914H, 6420		IA32_PMC_GP5_CTR
Full Width Writable General Performance Counter 5 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 5 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1915H, 6421		IA32_PMC_GP5_CFG_A
IA32_PMC_GP5_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 5. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 5
Register Address: 1916H, 6422		IA32_PMC_GP5_CFG_B
IA32_PMC_GP5_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 1917H, 6423		IA32_PMC_GP5_CFG_C
IA32_PMC_GP5_CFG_C (R/W) Extended Perf event selector for GP counter 5. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[5] = 1 or CPUID.23H.02H:EAX[5] = 1
Register Address: 1918H, 6424		IA32_PMC_GP6_CTR
Full Width Writable General Performance Counter 6 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 6 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1919H, 6425		IA32_PMC_GP6_CFG_A
IA32_PMC_GP6_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 6. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 6
Register Address: 191AH, 6426		IA32_PMC_GP6_CFG_B

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
IA32_PMC_GP6_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 191BH, 6427		IA32_PMC_GP6_CFG_C
IA32_PMC_GP6_CFG_C (R/W) Extended Perf event selector for GP counter 6. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[6] = 1 or CPUID.23H.02H:EAX[6] = 1
Register Address: 191CH, 6428		IA32_PMC_GP7_CTR
Full Width Writable General Performance Counter 7 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 7 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 191DH, 6429		IA32_PMC_GP7_CFG_A
IA32_PMC_GP7_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 7. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 7
Register Address: 191EH, 6430		IA32_PMC_GP7_CFG_B
IA32_PMC_GP7_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 191FH, 6431		IA32_PMC_GP7_CFG_C
IA32_PMC_GP7_CFG_C (R/W) Extended Perf event selector for GP counter 7. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[7] = 1 or CPUID.23H.02H:EAX[7] = 1
Register Address: 1920H, 6432		IA32_PMC_GP8_CTR
Full Width Writable General Performance Counter 8 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 8 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1921H, 6433		IA32_PMC_GP8_CFG_A
IA32_PMC_GP8_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 8. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 8
Register Address: 1922H, 6430		IA32_PMC_GP8_CFG_B
IA32_PMC_GP8_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 1923H, 6431		IA32_PMC_GP8_CFG_C
IA32_PMC_GP8_CFG_C (R/W) Extended Perf event selector for GP counter 8. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[8] = 1 or CPUID.23H.02H:EAX[8] = 1
Register Address: 1924H, 6436		IA32_PMC_GP9_CTR

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Full Width Writable General Performance Counter 9 (R/W) See IA32_PMC_GP0_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 9 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1925H, 6437		IA32_PMC_GP9_CFG_A
IA32_PMC_GP9_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 9. See IA32_PMC_GP0_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 9
Register Address: 1926H, 6430		IA32_PMC_GP9_CFG_B
IA32_PMC_GP9_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 1927H, 6431		IA32_PMC_GP9_CFG_C
IA32_PMC_GP9_CFG_C (R/W) Extended Perf event selector for GP counter 9. See IA32_PMC_GP0_CFG_C (1903H) for reference; similar format.		CPUID.23H.05H:EAX[9] = 1 or CPUID.23H.02H:EAX[9] = 1
Register Address: 1980H, 6528		IA32_PMC_FX0_CTR
Fixed-Function Performance Counter 0 (R/W) Instructions retired.		If CPUID.0AH:EDX[4:0] > 0
47:0	FIXED_COUNTER Instructions Retired Counter.	
63:46	Reserved.	
Register Address: 1982H, 6530		IA32_PMC_FX0_CFG_B
Fixed-Function Counter Reload Configuration Register (R/W)		
1:0	Reserved.	
2	RELOAD_PMC2 Reload Fixed-Function Counter0 when GP2 overflows.	
3	RELOAD_PMC3 Reload Fixed-Function Counter0 when GP3 overflows.	
4	RELOAD_PMC4 Reload Fixed-Function Counter0 when GP4 overflows.	
5	RELOAD_PMC5 Reload Fixed-Function Counter0 when GP5 overflows.	
6	RELOAD_PMC6 Reload Fixed-Function Counter0 when GP6 overflows.	
7	RELOAD_PMC7 Reload Fixed-Function Counter0 when GP7 overflows.	
33:8	Reserved.	
32	RELOAD_FCO Reload Fixed-Function Counter0 when FCO overflows.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
33	RELOAD_FC1 Reload Fixed-Function Counter0 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of Fixed-Function Counter 0.	
63:49	Reserved.	
Register Address: 1983H, 6531		IA32_PMC_FX0_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 0 (R/W)		CPUID.23H.05H:ECX[0] = 1 or CPUID.23H.02H:EBX[0] = 1
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
34:32	Reserved.	
35	ALLOW_IN_RECORD (R/W) If 1, indicates that this counter's value will be included in the PEBS record if the associated CNTR subgroup is enabled.	
38:36	CNTR (R/W) When any bit is set, the counter group will be included in the PEBS record. - Bit 36: If 1, the general-purpose counters subgroup will be included. - Bit 37: If 1, the fixed-function counters subgroup will be included. - Bit 38: If 1, the performance metrics subgroup will be included and will be reset	
39	Reserved.	
41:40	LBR (R/W) Identifies the number of Last Branch Records (LBRs) to record. The encodings of this field are as follows: 00 - LBR group will not be recorded. The group is disabled. 01 - 8 LBRs will be recorded. 10 - 16 LBRs will be recorded. 11 - The number of LBRs to be recorded is IA32_LBR_DEPTH.DEPTH.	
48:42	Reserved.	
55:49	XER (R/W) When set, enables particular XSAVE-Enabled Registers to be included in the PEBS record. The encodings for the register subgroup are: - Bit 49 - SSER: Record XMM0-XMM15. - Bit 50 - YMMHIR: Record the upper 128 bits of YMM0-YMM15. - Bit 51 - EGPR: Record R16-R31. - Bit 52 - Reserved and must be zero. - Bit 53 - OPMASKR: Record K0-K7. - Bit 54 - ZMMHIR: Record the upper 256 bits of ZMM0-ZMM15. - Bit 55 - Hi16ZMMR: Record ZMM16-ZMM31.	
60:56	Reserved.	
61	GPR (R/W) If 1, enables the General-Purpose Registers Group to be included in the PEBS record.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
62	AUX (R/W) If 1, enables the Auxiliary Group to be included in the PEBS record.	
63	PEBS (R/W) If 1, PEBS is enabled for this counter.	
Register Address: 1984H, 6532		IA32_PMC_FX1_CTR
Fixed-Function Performance Counter 1 (R/W) Unhalted core clock cycles.		If CPUID.0AH:EDX[4:0] > 1
47:0	FIXED_COUNTER Unhalted core clock cycles counter.	
63:46	Reserved.	
Register Address: 1986H, 6534		IA32_PMC_FX1_CFG_B
Fixed-Function Counter Reload Configuration Register (R/W)		
1:0	Reserved.	
2	RELOAD_PMC2 Reload Fixed-Function Counter1 when GP2 overflows.	
3	RELOAD_PMC3 Reload Fixed-Function Counter1 when GP3 overflows.	
4	RELOAD_PMC4 Reload Fixed-Function Counter1 when GP4 overflows.	
5	RELOAD_PMC5 Reload Fixed-Function Counter1 when GP5 overflows.	
6	RELOAD_PMC6 Reload Fixed-Function Counter1 when GP6 overflows.	
7	RELOAD_PMC7 Reload Fixed-Function Counter1 when GP7 overflows.	
31:8	Reserved.	
32	RELOAD_FC0 Reload Fixed-Function Counter1 when FC0 overflows.	
33	RELOAD_FC1 Reload Fixed-Function Counter1 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of Fixed-Function Counter 1.	
63:49	Reserved.	
Register Address: 1987H, 6532		IA32_PMC_FX1_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 1 (R/W) See IA32_PMC_FX0_CFG_C (1983H) for reference; similar format.		CPUID.23H.05H:ECX[1] = 1 or CPUID.23H.02H:EBX[1] = 1
Register Address: 1988H, 6536		IA32_PMC_FX2_CTR
Fixed-Function Performance Counter 2 (R/W) Unhalted core reference cycles.		If CPUID.0AH:EDX[4:0] > 2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
47:0	FIXED_COUNTER Unhalted core reference cycles counter.	
63:48	Reserved.	
Register Address: 198AH, 6538		IA32_PMC_FX2_CFG_B
Fixed-Function Counter Reload Configuration Register (R/W)		
1:0	Reserved.	
2	RELOAD_PMC2 Reload Fixed-Function Counter2 when GP2 overflows.	
3	RELOAD_PMC3 Reload Fixed-Function Counter2 when GP3 overflows.	
4	RELOAD_PMC4 Reload Fixed-Function Counter2 when GP4 overflows.	
5	RELOAD_PMC5 Reload Fixed-Function Counter2 when GP5 overflows.	
6	RELOAD_PMC6 Reload Fixed-Function Counter2 when GP6 overflows.	
7	RELOAD_PMC7 Reload Fixed-Function Counter2 when GP7 overflows.	
31:8	Reserved.	
32	RELOAD_FC0 Reload Fixed-Function Counter2 when FC0 overflows.	
33	RELOAD_FC1 Reload Fixed-Function Counter2 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of Fixed-Function Counter 2.	
63:49	Reserved.	
Register Address: 198BH, 6539		IA32_PMC_FX2_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 2 (R/W) See IA32_PMC_FX0_CFG_C (1983H) for reference; similar format.		CPUID.23H.05H:ECX[2] = 1 or CPUID.23H.02H:EBX[2] = 1
Register Address: 198CH, 6540		IA32_PMC_FX3_CTR
Fixed-Function Performance Counter 3 (R/W) Top-down Microarchitecture Analysis unhalted number of available slots.		If CPUID.0AH:EDX[4:0] > 3
47:0	FIXED_COUNTER Top-down microarchitecture analysis unhalted number of available slots counter.	
63:48	Reserved.	
Register Address: 198EH, 6542		IA32_PMC_FX3_CFG_B
Fixed-Function Counter Reload Configuration Register (R/W)		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
1:0	Reserved.	
2	RELOAD_PMC2 Reload Fixed-Function Counter3 when GP2 overflows.	
3	RELOAD_PMC3 Reload Fixed-Function Counter3 when GP3 overflows.	
4	RELOAD_PMC4 Reload Fixed-Function Counter3 when GP4 overflows.	
5	RELOAD_PMC5 Reload Fixed-Function Counter3 when GP5 overflows.	
6	RELOAD_PMC6 Reload Fixed-Function Counter3 when GP6 overflows.	
7	RELOAD_PMC7 Reload Fixed-Function Counter3 when GP7 overflows.	
31:8	Reserved.	
32	RELOAD_FC0 Reload Fixed-Function Counter3 when FC0 overflows.	
33	RELOAD_FC1 Reload Fixed-Function Counter3 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of Fixed-Function Counter 3.	
63:49	Reserved.	
Register Address: 198FH, 6543		IA32_PMC_FX3_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 3 (R/W) See IA32_PMC_FX0_CFG_C (1983H) for reference; similar format.		CPUID.23H.05H:ECX[3] = 1 or CPUID.23H.02H:EBX[3] = 1
Register Address: 1990H, 6544		IA32_PMC_FX4_CTR
Fixed-Function Performance Counter 4 (R/W) Top-down bad speculation.		If CPUID.0AH:EDX[4:0] > 4
47:0	FIXED_COUNTER Top-down bad speculation counter.	
63:48	Reserved.	
Register Address: 1993H, 6547		IA32_PMC_FX4_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 4 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	
Register Address: 1994H, 6548		IA32_PMC_FX5_CTR
Fixed-Function Performance Counter 5 (R/W) Top-down frontend bound.		If CPUID.0AH:EDX[4:0] > 5

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
47:0	FIXED_COUNTER Top-down frontend-bound counter.	
63:48	Reserved.	
Register Address: 1997H, 6551		IA32_PMC_FX5_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 5 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	
Register Address: 1998H, 6552		IA32_PMC_FX6_CTR
Fixed-Function Performance Counter 6 (R/W) Top-down retiring.		If CPUID.0AH:EDX[4:0] > 6
47:0	FIXED_COUNTER Top-down retiring counter.	
63:48	Reserved.	
Register Address: 199BH, 6555		IA32_PMC_FX6_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 6 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	
Register Address: 1B01H, 6913		IA32_UARCH_MISC_CTL
IA32_UARCH_MISC_CTL (R/W)		If IA32_ARCH_CAPABILITIES[12] = 1
0	Data Operand Independent Timing Mode (DOITM).	If IA32_ARCH_CAPABILITIES[12] = 1
63:1	Reserved.	
Register Address: 4000_0000H–4000_00FFH		Reserved MSR Address Space
All existing and future processors will not implement MSRs in this range.		
Register Address: C000_0080H		IA32_EFER
Extended Feature Enables		If (CPUID.80000001H:EDX[20] CPUID.80000001H:EDX[29])
0	SYSCALL Enable: IA32_EFER.SCE (R/W) Enables SYSCALL/SYSRET instructions in 64-bit mode.	
7:1	Reserved.	
8	IA-32e Mode Enable: IA32_EFER.LME (R/W) Enables IA-32e mode operation.	
9	Reserved.	
10	IA-32e Mode Active: IA32_EFER.LMA (R) Indicates IA-32e mode is active when set.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
11	Execute Disable Bit Enable: IA32_EFER.NXE (R/W)	
63:12	Reserved.	
Register Address: C000_0081H		IA32_STAR
System Call Target Address (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0082H		IA32_LSTAR
IA-32e Mode System Call Target Address (R/W) Target RIP for the called procedure when SYSCALL is executed in 64-bit mode.		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0083H		IA32_CSTAR
IA-32e Mode System Call Target Address (R/W) Not used, as the SYSCALL instruction is not recognized in compatibility mode.		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0084H		IA32_FMASK
System Call Flag Mask (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0100H		IA32_FS_BASE
Map of BASE Address of FS (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0101H		IA32_GS_BASE
Map of BASE Address of GS (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0102H		IA32_KERNEL_GS_BASE
Swap Target of BASE Address of GS (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0103H		IA32_TSC_AUX
Auxiliary TSC (R/W)		If CPUID.80000001H:EDX[27] = 1 or CPUID.07H.00H:ECX[22] = 1
31:0	AUX: Auxiliary signature of TSC.	
63:32	Reserved.	

NOTES:

1. Some older processors may have supported this MSR as model-specific and do not enumerate it with CPUID.
2. In processors based on Intel NetBurst® microarchitecture, MSR addresses 180H-197H are supported, software must treat them as model-specific. Starting with Intel Core Duo processors, MSR addresses 180H-185H, 188H-197H are reserved.
3. The *_ADDR MSRs may or may not be present; this depends on flag settings in IA32_MCi_STATUS. See Section 18.3.2.3 and Section 18.3.2.4 for more information.
4. Further details on Key Locker and usage of this MSR can be found here:

<https://software.intel.com/content/www/us/en/develop/download/intel-key-locker-specification.html>.

2.2 MSRS IN THE INTEL® CORE™ 2 PROCESSOR FAMILY

Table 2-3 lists model-specific registers (MSRs) for the Intel Core 2 processor family and for Intel Xeon processors based on Intel Core microarchitecture, architectural MSR addresses are also included in Table 2-3. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_0FH, see Table 2-1.

MSRs listed in Table 2-2 and Table 2-3 are also supported by processors based on the Enhanced Intel Core microarchitecture. Processors based on the Enhanced Intel Core microarchitecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_17H.

The column “Shared/Unique” applies to multi-core processors based on Intel Core microarchitecture. “Unique” means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. “Shared” means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, “MSRs in Pentium Processors.”		Unique
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, “MSRs in Pentium Processors.”		Unique
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, “Time-Stamp Counter,” and Table 2-2.		Unique
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Shared
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Shared
7:0	Reserved.	
12:8	Maximum Qualified Ratio (R) The maximum allowed bus ratio.	
49:13	Reserved.	
52:50	See Table 2-2.	
63:53	Reserved.	
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, “Local APIC Status and Location,” and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
0	Reserved.	
1	Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
2	Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
3	MCERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
4	Address Parity Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
5	Reserved.	
6	Reserved.	
7	BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
8	Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled.	
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.	
10	MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	
11	Intel TXT Capable Chipset. (R/O) 1 = Present; 0 = Not Present.	
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	
13	Reserved.	
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes.	
15	Reserved.	
17:16	APIC Cluster ID (R/O)	
18	N/2 Non-Integer Bus Ratio (R/O) 0 = Integer ratio; 1 = Non-integer ratio.	
19	Reserved.	
21: 20	Symmetric Arbitration ID (R/O)	
26:22	Integer Bus Frequency Ratio (R/O)	
Register Address: 3AH, 58	MSR_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Unique
3	SMRR Enable (R/WL) When this bit is set and the lock bit is set, this makes the SMRR_PHYS_BASE and SMRR_PHYS_MASK registers read visible and writeable while in SMM.	Unique
Register Address: 40H, 64	MSR_LASTBRANCH_0_FROM_IP	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Last Branch Record 0 From IP (R/W) One of four pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 20.5.		Unique
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of four pairs of last branch record registers on the last branch record stack. This To_IP part of the stack contains pointers to the destination instruction.		Unique
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Unique
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Unique
Register Address: A0H, 160	MSR_SMRR_PHYSBASE	
System Management Mode Base Address register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.		Unique
11:0	Reserved.	
31:12	PhysBase: SMRR physical Base Address.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
63:32	Reserved.	
Register Address: A1H, 161	MSR_SMRR_PHYSMASK	
System Management Mode Physical Address Mask register (wO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.		Unique
10:0	Reserved.	
11	Valid: Physical address base and range mask are valid.	
31:12	PhysMask: SMRR physical address range mask.	
63:32	Reserved.	
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Unique
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Unique
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Intel Core microarchitecture.		Shared
2:0	▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333)	
	133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B. 266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B.	
63:3	Reserved.	
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Enhanced Intel Core microarchitecture.		Shared

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
2:0	▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333) ▪ 110B: 400 MHz (FSB 1600) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B. 266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 110B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 111B.	
63:3	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Unique
11	SMRR Capability Using MSR 0A0H and 0A1H (R)	Unique
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSELO	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Shared
Register Address: 198H, 408	MSR_PERF_STATUS	
Current performance status. See Section 17.1.1, "Software Interface For Initiating Performance State Transitions."		Shared
15:0	Current Performance State Value	
30:16	Reserved.	
31	XE Operation (R/O). If set, XE operation is enabled. Default is cleared.	
39:32	Reserved.	
44:40	Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.	
45	Reserved.	
46	Non-Integer Bus Ratio (R/O) Indicates non-integer bus ratio is enabled. Applies processors based on Enhanced Intel Core microarchitecture.	
63:47	Reserved.	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Thermal Interrupt Control (R/W) See Table 2-2.		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Unique
15:0	Reserved.	
16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle). 1 = Thermal Monitor 2 (thermally-initiated frequency transitions). If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.	
63:16	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
8	Reserved.	
9	Hardware Prefetcher Disable (R/W) When set, disables the hardware prefetcher operation on streams of data. When clear (default), enables the prefetch queue. Disabling of the hardware prefetcher may impact processor performance.	
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Shared
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.01H:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.	Shared
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	Adjacent Cache Line Prefetch Disable (R/W) When set to 1, the processor fetches the cache line that contains data currently required by the processor. When set to 0, the processor fetches cache lines that comprise a cache line pair (128 bytes). Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.	Shared
20	Enhanced Intel SpeedStep Technology Select Lock (R/W0) When set, this bit causes the following bits to become read-only: - Enhanced Intel SpeedStep Technology Select Lock (this bit). - Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.	Shared
21	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Shared
23	xTPR Message Disable (R/W) See Table 2-2.	Shared
33:24	Reserved.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
34	XD Bit Disable (R/W) When set to 1, the Execute Disable Bit feature (XD Bit) is disabled and the XD Bit extended feature flag will be clear (CPUID.80000001H:EDX[20] = 0). When set to a 0 (default), the Execute Disable Bit feature (if available) allows the OS to enable PAE paging and take advantage of data only pages. BIOS must not alter the contents of this bit location if XD bit is not supported. Writing this bit to 1 when the XD Bit extended feature flag is set to 0 may generate a #GP exception.	Unique
36:35	Reserved.	
37	DCU Prefetcher Disable (R/W) When set to 1, the DCU L1 data cache prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The DCU prefetcher is an L1 data cache prefetcher. When the DCU prefetcher detects multiple loads from the same line done within a time limit, the DCU prefetcher assumes the next line will be required. The next line is prefetched in to the L1 data cache from memory or L2.	Unique
38	IDA Disable (R/W) When set to 1 on processors that support IDA, the Intel Dynamic Acceleration feature (IDA) is disabled and the IDA_Enable feature flag will be cleared (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of IDA is enabled. Note: The power-on default value is used by BIOS to detect hardware support of IDA. If the power-on default value is 1, IDA is available in the processor. If the power-on default value is 0, IDA is not available.	Shared
39	IP Prefetcher Disable (R/W) When set to 1, the IP prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The IP prefetcher is an L1 data cache prefetcher. The IP prefetcher looks for sequential load history to determine whether to prefetch the next expected data into the L1 cache from memory or L2.	Unique
63:40	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Unique
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Unique
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Unique
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Unique
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Unique
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Unique
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Unique
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Unique
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Unique
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Unique
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Unique
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Unique
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Unique
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Unique
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Unique
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2.		Unique
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Unique
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Unique
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Unique
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Unique
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Unique
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Unique
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Unique
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Unique
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Unique
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Unique
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Unique
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Unique
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Unique
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Unique
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Unique

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Unique
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 20.4.1, "IA32_DEBUGCTL MSR."		Unique
Register Address: 345H, 837	MSR_PERF_CAPABILITIES	
R/O. This applies to processors that do not support architectural PerfMon version 2.		Unique
5:0	LBR Format. See Table 2-2.	
6	PEBS Record Format.	
7	PEBSSaveArchRegs. See Table 2-2.	
63:8	Reserved.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Unique
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38EH, 910	MSR_PERF_GLOBAL_STATUS	
See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	MSR_PERF_GLOBAL_CTRL	
See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	MSR_PERF_GLOBAL_OVF_CTRL	
See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Unique
0	Enable PEBS on IA32_PMC0. (R/W)	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 402H, 1026	IA32_MCO_ADDR	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Unique
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Unique
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Unique
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Unique
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 40CH, 1036	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Unique
Register Address: 40DH, 1037	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Unique
Register Address: 40EH, 1038	IA32_MC4_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 410H, 1040	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		
Register Address: 411H, 1041	IA32_MC3_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		
Register Address: 412H, 1042	IA32_MC3_ADDR	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 413H, 1043	IA32_MC3_MISC	
Machine Check Error Reporting Register: Contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 414H, 1044	IA32_MC5_CTL	
Machine Check Error Reporting Register: Controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).		Unique
Register Address: 415H, 1045	IA32_MC5_STATUS	
Machine Check Error Reporting Register: Contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.		Unique
Register Address: 416H, 1046	IA32_MC5_ADDR	
Machine Check Error Reporting Register: Contains the address of the code or data memory location that produced the machine-check error if the ADDR_V flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 417H, 1047	IA32_MC5_MISC	
Machine Check Error Reporting Register: Contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 419H, 1045	IA32_MC6_STATUS	
Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 26.		Unique
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL52	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2. See Section 22.6.3.4, "Debug Store (DS) Mechanism."		Unique
Register Address: 107CCH, 67532	MSR_EMON_L3_CTR_CTL0	
GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107CDH, 67533	MSR_EMON_L3_CTR_CTL1	
GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107CEH, 67534	MSR_EMON_L3_CTR_CTL2	
GSPNQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107CFH, 67535	MSR_EMON_L3_CTR_CTL3	
GSPNQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107D0H, 67536	MSR_EMON_L3_CTR_CTL4	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107D1H, 67537	MSR_EMON_L3_CTR_CTL5	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107D2H, 67538	MSR_EMON_L3_CTR_CTL6	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107D3H, 67539	MSR_EMON_L3_CTR_CTL7	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: 107D8H, 67544	MSR_EMON_L3_GL_CTL	
L3/FSB Common Control Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 20.2.2.		Unique
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Unique
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Unique
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Unique
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Unique

2.3 MSRS IN THE 45 NM AND 32 NM INTEL ATOM® PROCESSOR FAMILY

Table 2-4 lists model-specific registers (MSRs) for 45 nm and 32 nm Intel Atom processors, architectural MSR addresses are also included in Table 2-4. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_1CH, 06_26H, 06_27H, 06_35H, or 06_36H; see Table 2-1.

The column “Shared/Unique” applies to logical processors sharing the same core in processors based on the Intel Atom microarchitecture. “Unique” means each logical processor has a separate MSR, or a bit field in an MSR governs only a logical processor. “Shared” means the MSR or the bit field in an MSR address governs the operation of both logical processors in the same core.

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, “MSRs in Pentium Processors.”		Shared
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, “MSRs in Pentium Processors.”		Shared
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, “Time-Stamp Counter,” and see Table 2-2.		Unique
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Shared
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Shared
7:0	Reserved.	
12:8	Maximum Qualified Ratio (R) The maximum allowed bus ratio.	
63:13	Reserved.	
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, “Local APIC Status and Location,” and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
0	Reserved.	
1	Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
2	Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
3	AERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
4	BERR# Enable for initiator bus requests (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
5	Reserved.	
6	Reserved.	
7	BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
8	Reserved.	
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.	
10	AERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0.	
11	Reserved.	
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0.	
13	Reserved.	
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes.	
15	Reserved.	
17:16	APIC Cluster ID (R/O) Always 00B.	
19: 18	Reserved.	
21: 20	Symmetric Arbitration ID (R/O) Always 00B.	
26:22	Integer Bus Frequency Ratio (R/O)	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Unique
Register Address: 40H, 64	MSR_LASTBRANCH_O_FROM_IP	
Last Branch Record 0 From IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.5.		Unique

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 44H, 68	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 45H, 69	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 46H, 70	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 47H, 71	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.		Unique
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 64H, 100	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 65H, 101	MSR_LASTBRANCH_5_TO_IP	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 66H, 102	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 67H, 103	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Shared
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Unique
Register Address: C1H, 193	IA32_PMC0	
Performance counter register See Table 2-2.		Unique
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Unique
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Intel Atom microarchitecture.		Shared
2:0	111B: 083 MHz (FSB 333) 101B: 100 MHz (FSB 400) 001B: 133 MHz (FSB 533) 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.	
63:3	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: FEH, 254	IA32_MTRRCAP	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Memory Type Range Register (R) See Table 2-2.		Shared
Register Address: 11EH, 281	MSR_BBL_CR_CTL3	
Control Register 3 Used to configure the L2 Cache.		Shared
0	L2 Hardware Enabled (R/O) 1 = Indicates the L2 is hardware-enabled. 0 = Indicates the L2 is hardware-disabled.	
7:1	Reserved.	
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set, the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.	
22:9	Reserved.	
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.	
63:24	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSELO	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Shared
Register Address: 198H, 408	MSR_PERF_STATUS	
Performance Status		Shared
15:0	Current Performance State Value.	
39:16	Reserved.	
44:40	Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.	
63:45	Reserved.	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Shared
15:0	Reserved.	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle). 1 = Thermal Monitor 2 (thermally-initiated frequency transitions). If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.	
63:17	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		Unique
0	Fast-Strings Enable See Table 2-2.	
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
8	Reserved.	
9	Reserved.	
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Shared
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is cleared (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.01H:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.	Shared

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	Reserved.	
20	Enhanced Intel SpeedStep Technology Select Lock (R/WO) When set, this bit causes the following bits to become read-only: - Enhanced Intel SpeedStep Technology Select Lock (this bit). - Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.	Shared
21	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Unique
23	xTPR Message Disable (R/W) See Table 2-2.	Shared
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Unique
63:35	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Unique
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Shared
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2.		Shared
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Shared
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Shared
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Shared
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Shared
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Shared
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Shared
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Shared
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Shared
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Shared
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Shared
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Shared
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Shared
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Shared
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Shared
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Shared
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Shared
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Shared
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Shared

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Shared
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Shared
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Shared
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Shared
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Shared
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Shared
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Shared
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Unique
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Unique
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Unique
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Unique
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 20.4.1, "IA32_DEBUGCTL MSR."		Shared
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Unique
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 3F1H, 1009	IA32_PEBB_ENABLE (MSR_PEBB_ENABLE)	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Unique
0	Enable PEBS on IA32_PMC0 (R/W)	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Shared
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Shared
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Shared
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Shared
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Shared
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Shared
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Shared
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Shared
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Shared
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Shared
Register Address: 412H, 1042	IA32_MC4_ADDR	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2. See Section 22.6.3.4, "Debug Store (DS) Mechanism."		Unique
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Unique
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Unique
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Unique
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Unique

Table 2-5 lists model-specific registers (MSRs) that are specific to Intel Atom® processor with a CPUID Signature DisplayFamily_DisplayModel value of 06_27H.

Table 2-5. MSRs Supported by Intel Atom® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_27H

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3F8H, 1016	MSR_PKG_C2_RESIDENCY	
Package C2 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package

Table 2-5. MSRs Supported by Intel Atom® Processors (Contd.)with a CPUID Signature DisplayFamily_DisplayModel Value of 06_27H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:0	Package C2 Residency Counter (R/O) Time that this package is in processor-specific C2 states since last reset. Counts at 1 Mhz frequency.	Package
Register Address: 3F9H, 1017	MSR_PKG_C4_RESIDENCY	
Package C4 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C4 Residency Counter. (R/O) Time that this package is in processor-specific C4 states since last reset. Counts at 1 Mhz frequency.	Package
Register Address: 3FAH, 1018	MSR_PKG_C6_RESIDENCY	
Package C6 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter. (R/O) Time that this package is in processor-specific C6 states since last reset. Counts at 1 Mhz frequency.	Package

2.4 MSRS IN INTEL PROCESSORS BASED ON SILVERMONT MICROARCHITECTURE

Table 2-6 lists model-specific registers (MSRs) common to Intel processors based on the Silvermont microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_37H, 06_4AH, 06_4DH, 06_5AH, or 06_5DH; see Table 2-1. The MSRs listed in Table 2-6 are also common to processors based on the Airmont microarchitecture and newer microarchitectures for next generation Intel Atom processors.

Table 2-7 lists MSRs common to processors based on the Silvermont and Airmont microarchitectures, but not newer microarchitectures.

Table 2-8, Table 2-9, and Table 2-10 lists MSRs that are model-specific across processors based on the Silvermont microarchitecture.

In the Silvermont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Silvermont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, “MSRs in Pentium Processors.”		Module
Register Address: 1H, 1	IA32_P5_MC_TYPE	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Core
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, "Time-Stamp Counter," and Table 2-2.		Core
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, "Local APIC Status and Location," and Table 2-2.		Core
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Writes ignored.		Module
63:0	Reserved.	
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Core
31:0	SMI Count (R/O) Running count of SMI events since last RESET.	
63:32	Reserved.	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Core
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Core
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Core
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Module
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
18:16	C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit 10]: 100b - C4 is the max C-State to include 110b - C6 is the max C-State to include 111b - C7 is the max C-State to include	
63:19	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Core
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Core
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R) See Table 2-2.		Core
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction sets availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note: AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Core
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Core
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Core
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Core
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSELO	
See Table 2-2.		Core
7:0	Event Select	
15:8	UMask	
16	USR	
17	OS	
18	Edge	
19	PC	
20	INT	
21	Reserved.	
22	EN	
23	INV	
31:24	CMASK	
63:32	Reserved.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Core
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Module
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Core
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Core
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R) The default thermal throttling or PROCHOT# activation temperature in degrees C. The effective temperature for thermal throttling or PROCHOT# activation is "Temperature Target" + "Target Offset".	
29:24	Target Offset (R/W) Specifies an offset in degrees C to adjust the throttling and PROCHOT# activation temperature from the default target specified in TEMPERATURE_TARGET (bits 23:16).	
63:30	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
Offcore Response Event Select Register (R/W)		Module
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Module
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Core
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Core
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Core
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Core
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Core
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Core
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Core
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Core
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Core
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Core
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Core
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Core
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Core
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Core
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Core
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Core
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Core
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Core
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Core
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Core
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Core
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Core
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Core
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Core
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Core
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Core
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Core
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Core
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Core
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Core
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Core
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Core
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Core
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Core
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Core
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 20.4.1, "IA32_DEBUGCTL MSR."		Core
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Core
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Counts at the TSC Frequency.	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS."		Module
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Module
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS."		Module
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS."		Module
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Module
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS."		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Package
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Core
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Core
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Core
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Core
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Core
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Core
Register Address: 486H, 1158	IA32_VMX_CRO_FIXED0	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Core
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Core
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Core
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Core
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."		Core
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Core
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.		Core
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTL2	
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTL2	
Capability Reporting Register of Primary Processor-based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTL2	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2.		Core
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTL2	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2.		Core
Register Address: 491H, 1169	IA32_VMX_FMFUNC	
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Core
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Core
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 22.6.3.4, "Debug Store (DS) Mechanism."		Core
Register Address: 660H, 1632	MSR_CORE_C1_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C1 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C1 states. Counts at the TSC frequency.	
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.		Core
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Core
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Core
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Core
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Core
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Core
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Core
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Core
Register Address: C000_0103H	IA32_TSC_AUX	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
AUXILIARY TSC Signature (R/W) See Table 2-2.		Core

Table 2-7 lists model-specific registers (MSRs) that are common to Intel Atom® processors based on the Silvermont and Airmont microarchitectures but not newer microarchitectures.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Module
7:0	Reserved.	
13:8	Maximum Qualified Ratio (R) The maximum allowed bus ratio.	
49:13	Reserved.	
52:50	See Table 2-2.	
63:33	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Core
0	Lock (R/WL)	
1	Reserved.	
2	Enable VMX outside SMX operation (R/WL)	
Register Address: 40H, 64	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.5 and record format in Section 20.4.8.1.		Core
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 44H, 68	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 45H, 69	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 46H, 70	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 47H, 71	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.		Core
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 64H, 100	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 65H, 101	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 66H, 102	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 67H, 103	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information: Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * Scalable Bus Frequency.	Package
63:16	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Module
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-state support) 001b: C1 (Behavior is the same as 000b) 100b: C4 110b: C6 111b: C7 (Silvermont only)	
9:3	Reserved.	
10	I/O MWait Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWait instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: 11EH, 281	MSR_BBL_CR_CTL3	
Control Register 3 Used to configure the L2 Cache.		Module
0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.	
7:1	Reserved.	
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.	
22:9	Reserved.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.	
63:24	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Core
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.	Module
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Core
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Core
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Core
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Module
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Core
21:19	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Core
23	xTPR Message Disable (R/W) See Table 2-2.	Module
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Core
37:35	Reserved.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be cleared (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Module
63:39	Reserved.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 20.9.2, "Filtering of Last Branch Records."		Core
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
0	Enable PEBS for precise event on IA32_PMC0 (R/W)	
Register Address: 3FAH, 1018	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Counts at the TSC Frequency.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.	Module	
63:0	Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.	

2.4.1 MSRs with Model-Specific Behavior in the Silvermont Microarchitecture

Table 2-8 lists MSRs that are specific to the Intel Atom[®] processor E3000 Series (CUID Signature DisplayFamily_DisplayModel value of 06_37H) and Intel Atom processors (CUID Signature DisplayFamily_DisplayModel value of 06_4AH, 06_5AH, or 06_5DH).

Table 2-8. Specific MSRs Supported by Intel Atom[®] Processors with a CUID Signature DisplayFamily_DisplayModel Value of 06_37H, 06_4AH, 06_5AH, or 06_5DH

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Silvermont microarchitecture.	Module	
2:0	100B: 080.0 MHz 000B: 083.3 MHz 001B: 100.0 MHz 010B: 133.3 MHz 011B: 116.7 MHz	
63:3	Reserved.	
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
	Unit Multipliers used in RAPL Interfaces (R/O) See Section 17.10.1, "RAPL Interfaces."	Package
3:0	Power Units Power related information (in milliWatts) is based on the multiplier, 2^{PU} ; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliWatts increment.	
7:4	Reserved.	
12:8	Energy Status Units Energy related information (in microJoules) is based on the multiplier, 2^{ESU} ; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microJoules increment.	
15:13	Reserved.	
19:16	Time Unit The value is 0000b, indicating time unit is in one second.	
63:20	Reserved.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W)	Package	

Table 2-8. Specific MSRs Supported by Intel Atom® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H, 06_4AH, 06_5AH, or 06_5DH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:0	Package Power Limit #1 (R/W) See Section 17.10.3, "Package RAPL Domain," and MSR_RAPL_POWER_UNIT in Table 2-8.	
15	Enable Power Limit #1 (R/W) See Section 17.10.3, "Package RAPL Domain."	
16	Package Clamping Limitation #1 (R/W) See Section 17.10.3, "Package RAPL Domain."	
23:17	Time Window for Power Limit #1 (R/W) In unit of second. If 0 is specified in bits [23:17], defaults to 1 second window.	
63:24	Reserved.	
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 17.10.3, "Package RAPL Domain," and MSR_RAPL_POWER_UNIT in Table 2-8.		Package
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 17.10.4, "PP0/PP1 RAPL Domains," and MSR_RAPL_POWER_UNIT in Table 2-8.		Package

Table 2-9 lists model-specific registers (MSRs) that are specific to the Intel Atom® processor E3000 Series (CPUID Signature DisplayFamily_DisplayModel value of 06_37H).

Table 2-9. Specific MSRs Supported by the Intel Atom® Processor E3000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 668H, 1640	MSR_CC6_DEMOTION_POLICY_CONFIG	
Core C6 Demotion Policy Config MSR		Package
63:0	Controls per-core C6 demotion policy. Writing a value of 0 disables core level HW demotion policy.	
Register Address: 669H, 1641	MSR_MC6_DEMOTION_POLICY_CONFIG	
Module C6 Demotion Policy Config MSR		Package
63:0	Controls module (i.e., two cores sharing the second-level cache) C6 demotion policy. Writing a value of 0 disables module level HW demotion policy.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Module
63:0	Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.	

Table 2-10 lists model-specific registers (MSRs) that are specific to Intel Atom® processor C2000 Series (CPUID Signature DisplayFamily_DisplayModel value of 06_4DH).

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4DH

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	Reserved.	
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
63:3	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode (R/W)		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.	Package
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.	Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 17.10.1, "RAPL Interfaces."		Package
3:0	Power Units Power related information (in milliWatts) is based on the multiplier, 2^{PU} ; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliWatts increment.	
7:4	Reserved.	

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4DH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
12:8	Energy Status Units. Energy related information (in microJoules) is based on the multiplier, 2^{ESU} ; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microJoules increment.	
15:13	Reserved.	
19:16	Time Unit The value is 0000b, indicating time unit is in one second.	
63:20	Reserved.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 66EH, 1646	MSR_PKG_POWER_INFO	
PKG RAPL Parameter (R/O)		Package
14:0	Thermal Spec Power (R/O) The unsigned integer value is the equivalent of the thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.	
63:15	Reserved.	

2.4.2 MSRs in Intel Atom® Processors Based on Airmont Microarchitecture

Intel Atom processor X7-Z8000 and X5-Z8000 series are based on the Airmont microarchitecture. These processors support MSRs listed in Table 2-6, Table 2-7, Table 2-8, and Table 2-11. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_4CH; see Table 2-1.

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Airmont microarchitecture.		Module
3:0	▪ 0000B: 083.3 MHz ▪ 0001B: 100.0 MHz ▪ 0010B: 133.3 MHz ▪ 0011B: 116.7 MHz ▪ 0100B: 080.0 MHz ▪ 0101B: 093.3 MHz ▪ 0110B: 090.0 MHz ▪ 0111B: 088.9 MHz ▪ 1000B: 087.5 MHz	
63:5	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Module
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: No limit 001b: C1 010b: C2 110b: C6 111b: C7	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/W0) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Module
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PMG_CST_CONFIG_CONTROL[bit 10]: 000b - C3 is the max C-State to include. 001b - Deep Power Down Technology is the max C-State. 010b - C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: 638H, 1592	MSR_PPO_POWER_LIMIT	
PPO RAPL Power Limit Control (R/W)		Package

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:0	PP0 Power Limit #1 (R/W) See Section 17.10.4, “PP0/PP1 RAPL Domains,” and MSR_RAPL_POWER_UNIT in Table 2-8.	
15	Enable Power Limit #1 (R/W) See Section 17.10.4, “PP0/PP1 RAPL Domains.”	
16	Reserved.	
23:17	Time Window for Power Limit #1 (R/W) Specifies the time duration over which the average power must remain below PP0_POWER_LIMIT #1(14:0). Supported Encodings: 0x0: 1 second time duration. 0x1: 5 second time duration (Default). 0x2: 10 second time duration. 0x3: 15 second time duration. 0x4: 20 second time duration. 0x5: 25 second time duration. 0x6: 30 second time duration. 0x7: 35 second time duration. 0x8: 40 second time duration. 0x9: 45 second time duration. 0xA: 50 second time duration. 0xB-0x7F - reserved.	
63:24	Reserved.	

2.5 MSRS IN INTEL ATOM® PROCESSORS BASED ON GOLDMONT MICROARCHITECTURE

Intel Atom processors based on the Goldmont microarchitecture support MSRs listed in Table 2-6 and Table 2-12. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_5CH; see Table 2-1.

In the Goldmont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Goldmont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Module
49:0	Reserved.	
52:50	See Table 2-2.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:33	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Core
0	Lock (R/WL)	
1	Enable VMX inside SMX operation (R/WL)	
2	Enable VMX outside SMX operation (R/WL)	
14:8	SENTER local functions enables (R/WL)	
15	SENTER global functions enable (R/WL)	
18	SGX global functions enable (R/WL)	
63:19	Reserved.	
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Core TSC ADJUST (R/W) See Table 2-2.		Core
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Core
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
30	Programmable TJ OFFSET (R/O) When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify a temperature offset.	Package
39:31	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: No limit 0001b: C1 0010b: C3 0011b: C6 0100b: C7 0101b: C7S 0110b: C8 0111b: C9 1000b: C10	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability enhancement. Accessible only while in SMM.		Core
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.	
59	Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.	
63:60	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 188H, 392	IA32_PERFEVTSEL2	
See Table 2-2.		Core
Register Address: 189H, 393	IA32_PERFEVTSEL3	
See Table 2-2.		Core
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Core
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.	Package
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Core
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Core
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Core
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Core
21:19	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Core
23	xTPR Message Disable (R/W) See Table 2-2.	Package
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Core
37:35	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Package
63:39	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	Reserved.	
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
63:3	Reserved.	
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org.		Package
0	EIST Hardware Coordination Disable (R/W) When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores. When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.	
21:1	Reserved.	
22	Thermal Interrupt Coordination Enable (R/W) If set, then thermal interrupt on one core is routed to all cores.	
63:23	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode by Core Groups (R/W) Specifies Maximum Ratio Limit for each Core Group. Max ratio for groups with more cores must decrease monotonically. For groups with less than 4 cores, the max ratio must be 32 or less. For groups with 4-5 cores, the max ratio must be 22 or less. For groups with more than 5 cores, the max ratio must be 16 or less.		Package
7:0	Maximum Ratio Limit for Active Cores in Group 0 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 0 threshold.	Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
15:8	Maximum Ratio Limit for Active Cores in Group 1 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 1 threshold, and greater than the Group 0 threshold.	Package
23:16	Maximum Ratio Limit for Active Cores in Group 2 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 2 threshold, and greater than the Group 1 threshold.	Package
31:24	Maximum Ratio Limit for Active Cores in Group 3 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 3 threshold, and greater than the Group 2 threshold.	Package
39:32	Maximum Ratio Limit for Active Cores in Group 4 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 4 threshold, and greater than the Group 3 threshold.	Package
47:40	Maximum Ratio Limit for Active Cores in Group 5 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 5 threshold, and greater than the Group 4 threshold.	Package
55:48	Maximum Ratio Limit for Active Cores in Group 6 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 6 threshold, and greater than the Group 5 threshold.	Package
63:56	Maximum Ratio Limit for Active Cores in Group 7 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 7 threshold, and greater than the Group 6 threshold.	Package
Register Address: 1AEH, 430	MSR_TURBO_GROUP_CORECNT	
Group Size of Active Cores for Turbo Mode Operation (R/W) Writes of 0 threshold is ignored.		Package
7:0	Group 0 Core Count Threshold Maximum number of active cores to operate under the Group 0 Max Turbo Ratio limit.	Package
15:8	Group 1 Core Count Threshold Maximum number of active cores to operate under the Group 1 Max Turbo Ratio limit. Must be greater than the Group 0 Core Count.	Package
23:16	Group 2 Core Count Threshold Maximum number of active cores to operate under the Group 2 Max Turbo Ratio limit. Must be greater than the Group 1 Core Count.	Package
31:24	Group 3 Core Count Threshold Maximum number of active cores to operate under the Group 3 Max Turbo Ratio limit. Must be greater than the Group 2 Core Count.	Package
39:32	Group 4 Core Count Threshold Maximum number of active cores to operate under the Group 4 Max Turbo Ratio limit. Must be greater than the Group 3 Core Count.	Package
47:40	Group 5 Core Count Threshold Maximum number of active cores to operate under the Group 5 Max Turbo Ratio limit. Must be greater than the Group 4 Core Count.	Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
55:48	Group 6 Core Count Threshold Maximum number of active cores to operate under the Group 6 Max Turbo Ratio limit. Must be greater than the Group 5 Core Count.	Package
63:56	Group 7 Core Count Threshold Maximum number of active cores to operate under the Group 7 Max Turbo Ratio limit. Must be greater than the Group 6 Core Count, and not less than the total number of processor cores in the package. E.g., specify 255.	Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 20.9.2, "Filtering of Last Branch Records."		Core
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
9	EN_CALL_STACK	
63:10	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org .		Core
0	Reserved.	
1	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	Package
63:2	Reserved.	
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Core
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Core
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
See Table 2-2.		Core

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address:	IA32_MTRR_PHYSMASK9	
213H, 531	See Table 2-2.	Core
Register Address:	IA32_MC0_CTL2	
280H, 640	See Table 2-2.	Module
Register Address:	IA32_MC1_CTL2	
281H, 641	See Table 2-2.	Module
Register Address:	IA32_MC2_CTL2	
282H, 642	See Table 2-2.	Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Module
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 300H, 768	MSR_SGXOWNEREPPOCH0	
Lower 64 Bit CR_SGXOWNEREPPOCH (W) Writes do not update CR_SGXOWNEREPPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 301H, 769	MSR_SGXOWNEREPPOCH1	
Upper 64 Bit CR_SGXOWNEREPPOCH (W) Writes do not update CR_SGXOWNEREPPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 22.2.4, "Architectural Performance Monitoring Version 4."		Core
0	Ovf_PMC0	
1	Ovf_PMC1	
2	Ovf_PMC2	
3	Ovf_PMC3	
31:4	Reserved.	
32	Ovf_FixedCtr0	
33	Ovf_FixedCtr1	
34	Ovf_FixedCtr2	
54:35	Reserved.	
55	Trace_ToPA_PMI	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
57:56	Reserved.	
58	LBR_Frz	
59	CTR_Frz	
60	ASCI	
61	Ovf_Uncore	
62	Ovf_BufDSSAVE	
63	CondChgd	
Register Address: 390H, 912	IA32_PERF_GLOBAL_STATUS_RESET	
See Table 2-2 and Section 22.2.4, "Architectural Performance Monitoring Version 4."		Core
0	Set 1 to clear Ovf_PMC0.	
1	Set 1 to clear Ovf_PMC1.	
2	Set 1 to clear Ovf_PMC2.	
3	Set 1 to clear Ovf_PMC3.	
31:4	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	
33	Set 1 to clear Ovf_FixedCtr1.	
34	Set 1 to clear Ovf_FixedCtr2.	
54:35	Reserved.	
55	Set 1 to clear Trace_ToPA_PMI.	
57:56	Reserved.	
58	Set 1 to clear LBR_Frz.	
59	Set 1 to clear CTR_Frz.	
60	Set 1 to clear ASCI.	
61	Set 1 to clear Ovf_Uncore.	
62	Set 1 to clear Ovf_BufDSSAVE.	
63	Set 1 to clear CondChgd.	
Register Address: 391H, 913	IA32_PERF_GLOBAL_STATUS_SET	
See Table 2-2 and Section 22.2.4, "Architectural Performance Monitoring Version 4."		Core
0	Set 1 to cause Ovf_PMC0 = 1.	
1	Set 1 to cause Ovf_PMC1 = 1.	
2	Set 1 to cause Ovf_PMC2 = 1.	
3	Set 1 to cause Ovf_PMC3 = 1.	
31:4	Reserved.	
32	Set 1 to cause Ovf_FixedCtr0 = 1.	
33	Set 1 to cause Ovf_FixedCtr1 = 1.	
34	Set 1 to cause Ovf_FixedCtr2 = 1.	
54:35	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
55	Set 1 to cause Trace_ToPA_PMI = 1.	
57:56	Reserved.	
58	Set 1 to cause LBR_Frz = 1.	
59	Set 1 to cause CTR_Frz = 1.	
60	Set 1 to cause ASCI = 1.	
61	Set 1 to cause Ovf_Uncore.	
62	Set 1 to cause Ovf_BufDSSAVE.	
63	Reserved.	
Register Address: 392H, 914	IA32_PERF_GLOBAL_INUSE	
See Table 2-2.		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2 and Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
0	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0. (R/W)	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Module
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 4C3H, 1219	IA32_A_PMC2	
See Table 2-2.		Core
Register Address: 4C4H, 1220	IA32_A_PMC3	
See Table 2-2.		Core
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.		Package
0	Lock (SMM-RW) When set to '1' locks this register from further changes.	
1	Reserved.	
2	SMM_Code_Chk_En (SMM-RW) This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.	
63:3	Reserved.	
Register Address: 4E2H, 1250	MSR_SMM_DELAYED	
SMM Delayed (SMM-RO) Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.		Package
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a processor core of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 4E3H, 1251	MSR_SMM_BLOCKED	
SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a processor core of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is 0FFFH. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 500H, 1280	IA32_SGX_SVN_STATUS	
Status and SVN Threshold of SGX Support for ACM (R/O)		Core
0	Lock See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."	
15:1	Reserved.	
23:16	SGX_SVN_SINIT See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."	
63:24	Reserved.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W) See Table 2-2.		Core
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W) See Table 2-2.		Core
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Core
0	TraceEn	
1	CYCEn	
2	OS	
3	User	
6:4	Reserved, must be zero.	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	MTCEn	
10	TSCEn	
11	DisRETC	
12	Reserved, must be zero.	
13	BranchEn	
17:14	MTCFreq	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
18	Reserved, must be zero.	
22:19	CycThresh	
23	Reserved, must be zero.	
27:24	PSBFreq	
31:28	Reserved, must be zero.	
35:32	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Core
0	FilterEn Writes ignored.	
1	ContextEn Writes ignored.	
2	TriggerEn Writes ignored.	
3	Reserved	
4	Error (R/W)	
5	Stopped	
31:6	Reserved, must be zero.	
48:32	PacketByteCnt	
63:49	Reserved, must be zero.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Core
4:0	Reserved	
63:5	CR3[63:5] value to match.	
Register Address: 580H, 1408	IA32_RTIT_ADDR0_A	
Region 0 Start Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 581H, 1409	IA32_RTIT_ADDR0_B	
Region 0 End Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 582H, 1410	IA32_RTIT_ADDR1_A	
Region 1 Start Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 583H, 1411	IA32_RTIT_ADDR1_B	
Region 1 End Address (R/W)		Core
63:0	See Table 2-2.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 17.10.1, "RAPL Interfaces."		Package
3:0	Power Units Power related information (in Watts) is in unit of $1W/2^{PU}$; where PU is an unsigned integer represented by bits 3:0. Default value is 1000b, indicating power unit is in 3.9 milliWatts increment.	
7:4	Reserved.	
12:8	Energy Status Units Energy related information (in Joules) is in unit of $1Joule/2^{ESU}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 01110b, indicating energy unit is in 61 microjoules.	
15:13	Reserved.	
19:16	Time Unit Time related information (in seconds) is in unit of $1S/2^{TU}$; where TU is an unsigned integer represented by bits 19:16. Default value is 1010b, indicating power unit is in 0.977 millisecond.	
63:20	Reserved.	
Register Address: 60AH, 1546	MSR_PKG_C3_IRT_L	
Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C3 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60BH, 1547	MSR_PKG_C6_IRT_L1	
Package C6/C7S Interrupt Response Limit 1 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7S state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7S state.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60CH, 1548	MSR_PKGC_IRTL2	
Package C7 Interrupt Response Limit 2 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C2 Residency Counter (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W)		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:0	Thermal Spec Power (R/W) See Section 17.10.3, "Package RAPL Domain."	
15	Reserved.	
30:16	Minimum Power (R/W) See Section 17.10.3, "Package RAPL Domain."	
31	Reserved.	
46:32	Maximum Power (R/W) See Section 17.10.3, "Package RAPL Domain."	
47	Reserved.	
54:48	Maximum Time Window (R/W) Specified by $2^Y * (1.0 + Z/4.0) * \text{Time\_Unit}$, where "Y" is the unsigned integer value represented by bits 52:48, "Z" is an unsigned integer represented by bits 54:53. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.	
63:55	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 632H, 1586	MSR_PKG_C10_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C10 Residency Counter (R/O) Value since last reset that the entire SOC is in an S0i3 state. Count at the same frequency as the TSC.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	
PP1 Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
ConfigTDP Control (R/W)		Package
7:0	MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.	
30:8	Reserved.	
31	TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
2	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
3	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
8:4	Reserved.	
9	Core Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.	
10	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
11	Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.	
12	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
13	Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
14	Maximum Efficiency Frequency Status (R0) When set, frequency is reduced below the maximum efficiency frequency.	
15	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
18	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24:20	Reserved.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
30	Maximum Efficiency Frequency Log When set, indicates that the Maximum Efficiency Frequency Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:31	Reserved.	
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 0 From IP (R/W) One of 32 pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 20.6 and record format in Section 20.4.8.1.		Core
0:47	From Linear Address (R/W)	
62:48	Signed extension of bits 47:0.	
63	Mispred	
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 690H, 1680	MSR_LASTBRANCH_16_FROM_IP	
Last Branch Record 16 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 691H, 1681	MSR_LASTBRANCH_17_FROM_IP	
Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 692H, 1682	MSR_LASTBRANCH_18_FROM_IP	
Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 693H, 1683	MSR_LASTBRANCH_19_FROM_IP	
Last Branch Record 19 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 694H, 1684	MSR_LASTBRANCH_20_FROM_IP	
Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 695H, 1685	MSR_LASTBRANCH_21_FROM_IP	
Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 696H, 1686	MSR_LASTBRANCH_22_FROM_IP	
Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 697H, 1687	MSR_LASTBRANCH_23_FROM_IP	
Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 698H, 1688	MSR_LASTBRANCH_24_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 699H, 1689	MSR_LASTBRANCH_25_FROM_IP	
Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69AH, 1690	MSR_LASTBRANCH_26_FROM_IP	
Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69BH, 1691	MSR_LASTBRANCH_27_FROM_IP	
Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69CH, 1692	MSR_LASTBRANCH_28_FROM_IP	
Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69DH, 1693	MSR_LASTBRANCH_29_FROM_IP	
Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69EH, 1694	MSR_LASTBRANCH_30_FROM_IP	
Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69FH, 1695	MSR_LASTBRANCH_31_FROM_IP	
Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of 32 pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the Destination instruction and elapsed cycles from last LBR update. See Section 20.6.		Core
0:47	Target Linear Address (R/W)	
63:48	Elapsed cycles from last update to the LBR.	
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DOH, 1744	MSR_LASTBRANCH_16_TO_IP	
Last Branch Record 16 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D1H, 1745	MSR_LASTBRANCH_17_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D2H, 1746	MSR_LASTBRANCH_18_TO_IP	
Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D3H, 1747	MSR_LASTBRANCH_19_TO_IP	
Last Branch Record 19 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D4H, 1748	MSR_LASTBRANCH_20_TO_IP	
Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D5H, 1749	MSR_LASTBRANCH_21_TO_IP	
Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D6H, 1750	MSR_LASTBRANCH_22_TO_IP	
Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D7H, 1751	MSR_LASTBRANCH_23_TO_IP	
Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D8H, 1752	MSR_LASTBRANCH_24_TO_IP	
Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D9H, 1753	MSR_LASTBRANCH_25_TO_IP	
Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DAH, 1754	MSR_LASTBRANCH_26_TO_IP	
Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DBH, 1755	MSR_LASTBRANCH_27_TO_IP	
Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DCH, 1756	MSR_LASTBRANCH_28_TO_IP	
Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DDH, 1757	MSR_LASTBRANCH_29_TO_IP	
Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DEH, 1758	MSR_LASTBRANCH_30_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DFH, 1759	MSR_LASTBRANCH_31_TO_IP	
Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID register (R/O)		Core
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version register (R/O)		Core
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority register (R/W)		Core
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority register (R/O)		Core
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI register (W/O)		Core
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination register (R/O)		Core
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector register (R/W)		Core
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service register bits [31:0] (R/O)		Core
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service register bits [63:32] (R/O)		Core
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service register bits [95:64] (R/O)		Core
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service register bits [127:96] (R/O)		Core
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service register bits [159:128] (R/O)		Core
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service register bits [191:160] (R/O)		Core
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service register bits [223:192] (R/O)		Core
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service register bits [255:224] (R/O)		Core
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode register bits [31:0] (R/O)		Core
Register Address: 819H, 2073	IA32_X2APIC_TMR1	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
x2APIC Trigger Mode register bits [63:32] (R/O)		Core
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode register bits [95:64] (R/O)		Core
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode register bits [127:96] (R/O)		Core
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode register bits [159:128] (R/O)		Core
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode register bits [191:160] (R/O)		Core
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode register bits [223:192] (R/O)		Core
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode register bits [255:224] (R/O)		Core
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request register bits [31:0] (R/O)		Core
Register Address: 821H, 2081	IA32_X2APIC_IRR1	
x2APIC Interrupt Request register bits [63:32] (R/O)		Core
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request register bits [95:64] (R/O)		Core
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request register bits [127:96] (R/O)		Core
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request register bits [159:128] (R/O)		Core
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request register bits [191:160] (R/O)		Core
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request register bits [223:192] (R/O)		Core
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request register bits [255:224] (R/O)		Core
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status register (R/W)		Core
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt register (R/W)		Core
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command register (R/W)		Core
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt register (R/W)		Core

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt register (R/W)		Core
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor register (R/W)		Core
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 register (R/W)		Core
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 register (R/W)		Core
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error register (R/W)		Core
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count register (R/W)		Core
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count register (R/O)		Core
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration register (R/W)		Core
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI register (W/O)		Core
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Core
31:0	Reserved.	
33:32	CLOS (R/W)	
63: 34	Reserved.	
Register Address: D10H, 3344	IA32_L2_QOS_MASK_0	
L2 Class Of Service Mask - CLOS 0 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 0.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	
Register Address: D11H, 3345	IA32_L2_QOS_MASK_1	
L2 Class Of Service Mask - CLOS 1 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 1.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	
Register Address: D12H, 3346	IA32_L2_QOS_MASK_2	
L2 Class Of Service Mask - CLOS 2 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 2.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D13H, 3347	IA32_L2_QOS_MASK_3	
L2 Class Of Service Mask - CLOS 3 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 3.		Package
0:19	CBM: Bit vector of available L2 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: D90H, 3472	IA32_BNDCFGS	
See Table 2-2.		Core
Register Address: DA0H, 3488	IA32_XSS	
See Table 2-2.		Core
See Table 2-6, and Table 2-12 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_5CH.		

2.6 MSRS IN INTEL ATOM® PROCESSORS BASED ON GOLDMONT PLUS MICROARCHITECTURE

Intel Atom processors based on the Goldmont Plus microarchitecture support MSRs listed in Table 2-6, Table 2-12, and Table 2-13. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_7AH; see Table 2-1. For an MSR listed in Table 2-13 that also appears in the model-specific tables of prior generations, Table 2-13 supersedes prior generation tables.

In the Goldmont Plus microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Goldmont Plus microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Core
0	Lock (R/WL)	
1	Enable VMX inside SMX operation (R/WL)	
2	Enable VMX outside SMX operation (R/WL)	
14:8	SENTER local functions enables (R/WL)	
15	SENTER global functions enable (R/WL)	
17	SGX Launch Control Enable (R/WL) This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR. Valid if CPUID.07H.00H:ECX[30] = 1.	

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
18	SGX global functions enable (R/WL)	
63:19	Reserved.	
Register Address: 8CH, 140	IA32_SGXLEPUBKEYHASH0	
See Table 2-2.		Core
Register Address: 8DH, 141	IA32_SGXLEPUBKEYHASH1	
See Table 2-2.		Core
Register Address: 8EH, 142	IA32_SGXLEPUBKEYHASH2	
See Table 2-2.		Core
Register Address: 8FH, 143	IA32_SGXLEPUBKEYHASH3	
See Table 2-2.		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
(R/W) See Table 2-2. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
0	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0.	
1	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC1.	
2	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC2.	
3	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC3.	
31:4	Reserved.	
32	Enable PEBS trigger and recording for IA32_FIXED_CTR0.	
33	Enable PEBS trigger and recording for IA32_FIXED_CTR1.	
34	Enable PEBS trigger and recording for IA32_FIXED_CTR2.	
63:35	Reserved.	
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Core
0	TraceEn	
1	CYCEn	
2	OS	
3	User	
4	PwrEvtEn	
5	FUPonPTW	
6	FabricEn	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	MTCEn	
10	TSCEn	

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
11	DisRETC	
12	PTWEn	
13	BranchEn	
17:14	MTCFreq	
18	Reserved, must be zero.	
22:19	CycThresh	
23	Reserved, must be zero.	
27:24	PSBFreq	
31:28	Reserved, must be zero.	
35:32	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 680H, 1664	MSR_LASTBRANCH_O_FROM_IP	
Last Branch Record 0 From IP (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. The From_IP part of the stack contains pointers to the source instruction. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 20.7, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture.”		Core
Register Address: 681H–69FH, 1665–1695	MSR_LASTBRANCH_i_FROM_IP	
Last Branch Record <i>i</i> From IP (R/W) See description of MSR_LASTBRANCH_O_FROM_IP; <i>i</i> = 1-31.		Core
Register Address: 6C0H, 1728	MSR_LASTBRANCH_O_TO_IP	
Last Branch Record 0 To IP (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. The To_IP part of the stack contains pointers to the Destination instruction. See also: ▪ Section 20.7, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture.”		Core
Register Address: 6C1H–6DFH, 1729–1759	MSR_LASTBRANCH_i_TO_IP	
Last Branch Record <i>i</i> To IP (R/W) See description of MSR_LASTBRANCH_O_TO_IP; <i>i</i> = 1-31.		Core
Register Address: DCOH, 3520	MSR_LASTBRANCH_INFO_0	
Last Branch Record 0 Additional Information (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. This part of the stack contains flag and elapsed cycle information. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 20.9.1, “LBR Stack.”		Core
Register Address: DC1H, 3521	MSR_LASTBRANCH_INFO_1	
Last Branch Record 1 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DC2H, 3522	MSR_LASTBRANCH_INFO_2	
Last Branch Record 2 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC3H, 3523	MSR_LASTBRANCH_INFO_3	
Last Branch Record 3 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC4H, 3524	MSR_LASTBRANCH_INFO_4	
Last Branch Record 4 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC5H, 3525	MSR_LASTBRANCH_INFO_5	
Last Branch Record 5 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC6H, 3526	MSR_LASTBRANCH_INFO_6	
Last Branch Record 6 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC7H, 3527	MSR_LASTBRANCH_INFO_7	
Last Branch Record 7 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC8H, 3528	MSR_LASTBRANCH_INFO_8	
Last Branch Record 8 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC9H, 3529	MSR_LASTBRANCH_INFO_9	
Last Branch Record 9 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCAH, 3530	MSR_LASTBRANCH_INFO_10	
Last Branch Record 10 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCBH, 3531	MSR_LASTBRANCH_INFO_11	
Last Branch Record 11 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCCH, 3532	MSR_LASTBRANCH_INFO_12	
Last Branch Record 12 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCDH, 3533	MSR_LASTBRANCH_INFO_13	
Last Branch Record 13 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCEH, 3534	MSR_LASTBRANCH_INFO_14	
Last Branch Record 14 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DCFH, 3535	MSR_LASTBRANCH_INFO_15	
Last Branch Record 15 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDOH, 3536	MSR_LASTBRANCH_INFO_16	
Last Branch Record 16 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD1H, 3537	MSR_LASTBRANCH_INFO_17	
Last Branch Record 17 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD2H, 3538	MSR_LASTBRANCH_INFO_18	
Last Branch Record 18 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD3H, 3539	MSR_LASTBRANCH_INFO_19	
Last Branch Record 19 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD4H, 3520	MSR_LASTBRANCH_INFO_20	
Last Branch Record 20 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD5H, 3521	MSR_LASTBRANCH_INFO_21	
Last Branch Record 21 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD6H, 3522	MSR_LASTBRANCH_INFO_22	
Last Branch Record 22 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD7H, 3523	MSR_LASTBRANCH_INFO_23	
Last Branch Record 23 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD8H, 3524	MSR_LASTBRANCH_INFO_24	
Last Branch Record 24 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD9H, 3525	MSR_LASTBRANCH_INFO_25	
Last Branch Record 25 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDAH, 3526	MSR_LASTBRANCH_INFO_26	
Last Branch Record 26 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDBH, 3527	MSR_LASTBRANCH_INFO_27	
Last Branch Record 27 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DDCH, 3528	MSR_LASTBRANCH_INFO_28	
Last Branch Record 28 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDDH, 3529	MSR_LASTBRANCH_INFO_29	
Last Branch Record 29 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDEH, 3530	MSR_LASTBRANCH_INFO_30	
Last Branch Record 30 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDFH, 3531	MSR_LASTBRANCH_INFO_31	
Last Branch Record 31 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
See Table 2-6, Table 2-12, and Table 2-13 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_7AH.		

2.7 MSRS IN INTEL ATOM® PROCESSORS BASED ON TREMONT MICROARCHITECTURE

Processors based on the Tremont microarchitecture support MSRs listed in Table 2-6, Table 2-12, Table 2-13, and Table 2-14. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_86H, 06_96H, or 06_9CH; see Table 2-1. For an MSR listed in Table 2-14 that also appears in the model-specific tables of prior generations, Table 2-14 supersedes prior generation tables.

In the Tremont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Tremont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
28:0	Reserved.	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
30	Reserved.	
31	Reserved.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
IA32 Core Capabilities Register If CPUID.07H.00H:EDX[30] = 1.		Core
4:0	Reserved.	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL (MSR address 33H).	
63:6	Reserved.	
Register Address: 2A0H, 672	MSR_PRMRR_BASE_0	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE: PRMRR BASE Memory Type.	
3	CONFIGURED: PRMRR BASE Configured.	
11:4	Reserved.	
51:12	BASE: PRMRR Base Address.	
63:52	Reserved.	
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
(R/W) See Table 2-2. See Section 22.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
$n:0$	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMCx. The maximum value n can be determined from CPUID.0AH:EAX[15:8].	
31: $n+1$	Reserved.	
32: $m:32$	Enable PEBS trigger and recording for IA32_FIXED_CTRx. The maximum value m can be determined from CPUID.0AH:EDX[4:0].	
59:33+m	Reserved.	
60	Pend a PerfMon Interrupt (PMI) after each PEBS event.	
62:61	Specifies PEBS output destination. Encodings: 00B: DS Save Area. 01B: Intel PT trace output. Supported if IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16] and CPUID.07H.00H:EBX[25] are set. 10B: Reserved. 11B: Reserved.	
63	Reserved.	
Register Address: 1309H–130BH, 4873–4875	MSR_RELOAD_FIXED_CTRx	
Reload value for IA32_FIXED_CTRx (R/W)		
47:0	Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and FIXED_CTRx is overflowed.	
63:48	Reserved.	
Register Address: 14C1H–14C4H, 5313–5316	MSR_RELOAD_PMCx	

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Reload value for IA32_PMCx (R/W)		Core
47:0	Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and PMCx is overflowed.	
63:48	Reserved.	
See Table 2-6, Table 2-12, Table 2-13, and Table 2-14 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_86H.		

2.8 MSRS IN PROCESSORS BASED ON NEHALEM MICROARCHITECTURE

Table 2-15 lists model-specific registers (MSRs) that are common for Nehalem microarchitecture. These include the Intel Core i7 and i5 processor family. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, 06_1FH, or 06_2EH; see Table 2-1. Additional MSRs specific to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, or 06_1FH are listed in Table 2-16. Some MSRs listed in these tables are used by BIOS. More information about these MSR can be found at <http://biosbits.org>.

The column “Scope” represents the package/core/thread scope of individual bit field of an MSR. “Thread” means this bit field must be programmed on each logical processor independently. “Core” means the bit field must be programmed on each processor core independently, logical processors in the same core will be affected by change of this bit on the other logical processor in the same core. “Package” means the bit field must be programmed once for each physical package. Change of a bit filed with a package scope will affect all logical processors in that physical package.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, “MSRs in Pentium Processors.”		Thread
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, “MSRs in Pentium Processors.”		Thread
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, “Time-Stamp Counter,” and Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Package
49:0	Reserved.	
52:50	See Table 2-2.	
63:53	Reserved.	
Register Address: 1BH, 27	IA32_APIC_BASE	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 13.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O) Running count of SMI events since last RESET.	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Thread
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Thread
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. The invariant TSC frequency can be computed by multiplying this ratio by 133.33 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
29	Programmable TDC-TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDC and TDP Limits for Turbo mode are programmable. When set to 0, indicates TDC and TDP Limits for Turbo mode are not programmable.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 133.33MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-state support) 001b: C1 (Behavior is the same as 000b) 010b: C3 011b: C6 100b: C7 101b and 110b: Reserved 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
23:16	Reserved.	
24	Interrupt filtering enable (R/W) When set, processor cores in a deep C-State will wake only when the event message is destined for that core. When 0, all processor cores in a deep C-State will wake for an event message.	
25	C3 state auto demotion enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
26	C1 state auto demotion enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Core
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit 10]: 000b - C3 is the max C-State to include. 001b - C6 is the max C-State to include. 010b - C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Thread
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Thread
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
See Table 2-2.		Thread
7:0	Event Select	
15:8	UMask	
16	USR	
17	OS	
18	Edge	
19	PC	
20	INT	
21	AnyThread	
22	EN	
23	INV	
31:24	CMASK	
63:32	Reserved.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Thread
Register Address: 188H, 392	IA32_PERFEVTSEL2	
See Table 2-2.		Thread
Register Address: 189H, 393	IA32_PERFEVTSEL3	
See Table 2-2.		Thread
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Core
15:0	Current Performance State Value.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:16	Reserved.	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Thread
0	Reserved.	
3:1	On demand Clock Modulation Duty Cycle (R/W)	
4	On demand Clock Modulation Enable (R/W)	
63:5	Reserved.	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Thread
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.	Thread
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Thread
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Thread
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Thread
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM. (R/W) See Table 2-2.	Thread
21:19	Reserved.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	Limit CPUID Maxval (R/W) See Table 2-2.	Thread
23	xTPR Message Disable (R/W) See Table 2-2.	Thread
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Thread
37:35	Reserved.	
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Package
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Thread
15:0	Reserved.	
23:16	Temperature Target (R) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
63:24	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	L2 Adjacent Cache Line Prefetcher Disable (R/W) If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	Core
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
3	DCU IP Prefetcher Disable (R/W) If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	Core
63:4	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org .		
0	EIST Hardware Coordination Disable (R/W) When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores. When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.	Package
1	Energy/Performance Bias Enable (R/W) This bit makes the IA32_ENERGY_PERF_BIAS register (MSR 1B0h) visible to software with Ring 0 privileges. This bit's status (1 or 0) is also reflected by CPUID.06H:ECX[3].	Thread
63:2	Reserved.	
Register Address: 1ACH, 428	MSR_TURBO_POWER_CURRENT_LIMIT	
See http://biosbits.org .		
14:0	TDP Limit (R/W) TDP limit in 1/8 Watt granularity.	Package
15	TDP Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.	Package
30:16	TDC Limit (R/W) TDC limit in 1/8 Amp granularity.	Package
31	TDC Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.	Package
63:32	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record Filtering Select Register (R/W) See Section 20.9.2, "Filtering of Last Branch Records."		Core
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org .		Core
0	Reserved.	
1	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	Package
63:2	Reserved.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Thread
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Thread
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Thread
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Thread
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Thread
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Thread
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Thread
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Thread
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Thread
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Thread
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Thread
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Thread
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Thread
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Thread
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Thread
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Thread
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Thread
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Thread
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Thread
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
See Table 2-2.		Thread
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Thread
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Thread
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Thread
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Thread
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Thread
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Thread
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Thread
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Thread
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Thread
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Thread
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Thread
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Thread
Register Address: 280H, 640	IA32_MC0_CTL2	
See Table 2-2.		Package
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Package
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Core

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Core
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Thread
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 20.4.1, "IA32_DEBUGCTL MSR."		Thread
5:0	LBR Format See Table 2-2.	
6	PEBS Record Format	
7	PEBSSaveArchRegs See Table 2-2.	
11:8	PEBS_REC_FORMAT See Table 2-2.	
12	SMM_FREEZE See Table 2-2.	
63:13	Reserved.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Thread
Register Address: 38EH, 910	MSR_PERF_GLOBAL_STATUS	
Provides single-bit status used by software to query the overflow condition of each performance counter. (R/O)		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
61	UNC_Ovf Uncore overflowed if 1.	
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities."		Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 22.6.2.2, "Global Counter Control Facilities." Allows software to clear counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.		Thread
Register Address: 390H, 912	MSR_PERF_GLOBAL_OVF_CTRL	
(R/W)		Thread
61	CLR_UNC_Ovf Set 1 to clear UNC_Ovf.	
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 22.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
0	Enable PEBS on IA32_PMC0 (R/W)	
1	Enable PEBS on IA32_PMC1 (R/W)	
2	Enable PEBS on IA32_PMC2 (R/W)	
3	Enable PEBS on IA32_PMC3 (R/W)	
31:4	Reserved.	
32	Enable Load Latency on IA32_PMC0 (R/W)	
33	Enable Load Latency on IA32_PMC1 (R/W)	
34	Enable Load Latency on IA32_PMC2 (R/W)	
35	Enable Load Latency on IA32_PMC3 (R/W)	
63:36	Reserved.	
Register Address: 3F6H, 1014	MSR_PEBS_LD_LAT	
See Section 22.3.1.1.2, "Load Latency Performance Monitoring Facility."		Thread
15:0	Minimum threshold latency value of tagged load operation that will be counted. (R/W)	
63:36	Reserved.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:0	Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C7 Residency Counter (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Package
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Package
Register Address: 403H, 1027	IA32_MCO_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Package
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 407H, 1031	IA32_MC1_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 413H, 1043	IA32_MC4_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Core
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Core
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 19.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		Thread
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2 and Appendix A.4, "VM-Exit Controls."		Thread
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2 and Appendix A.5, "VM-Entry Controls."		Thread
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2 and Appendix A.6, "Miscellaneous Data."		Thread
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2 and Appendix A.9, "VMCS Enumeration."		Thread
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 22.6.3.4, "Debug Store (DS) Mechanism."		Thread
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - See Section 20.9.1 and record format in Section 20.4.8.1.		Thread
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.		Thread
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID Register (R/O)		Thread
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version Register (R/O)		Thread
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority Register (R/W)		Thread
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority Register (R/O)		Thread
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI Register (W/O)		Thread
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination Register (R/O)		Thread
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector Register (R/W)		Thread
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service Register Bits [31:0] (R/O)		Thread
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service Register Bits [63:32] (R/O)		Thread
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service Register Bits [95:64] (R/O)		Thread
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service Register Bits [127:96] (R/O)		Thread
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service Register Bits [159:128] (R/O)		Thread
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service Register Bits [191:160] (R/O)		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service Register Bits [223:192] (R/O)		Thread
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service Register Bits [255:224] (R/O)		Thread
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits [31:0] (R/O)		Thread
Register Address: 819H, 2073	IA32_X2APIC_TMR1	
x2APIC Trigger Mode Register Bits [63:32] (R/O)		Thread
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode Register Bits [95:64] (R/O)		Thread
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode Register Bits [127:96] (R/O)		Thread
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode Register Bits [159:128] (R/O)		Thread
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode Register Bits [191:160] (R/O)		Thread
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode Register Bits [223:192] (R/O)		Thread
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode Register Bits [255:224] (R/O)		Thread
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request Register Bits [31:0] (R/O)		Thread
Register Address: 821H, 2081	IA32_X2APIC_IRR1	
x2APIC Interrupt Request Register Bits [63:32] (R/O)		Thread
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request Register Bits [95:64] (R/O)		Thread
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request Register Bits [127:96] (R/O)		Thread
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request Register Bits [159:128] (R/O)		Thread
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request Register Bits [191:160] (R/O)		Thread
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request Register Bits [223:192] (R/O)		Thread
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request Register Bits [255:224] (R/O)		Thread
Register Address: 828H, 2088	IA32_X2APIC_ESR	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
x2APIC Error Status Register (R/W)		Thread
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt Register (R/W)		Thread
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		Thread
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		Thread
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt Register (R/W)		Thread
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor Register (R/W)		Thread
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		Thread
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 Register (R/W)		Thread
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error Register (R/W)		Thread
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count Register (R/W)		Thread
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count Register (R/O)		Thread
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration Register (R/W)		Thread
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI Register (W/O)		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 20.17.2, "IA32_TSC_AUX Register and RDTSCP Support."		Thread

2.8.1 Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

The Intel Xeon Processor 5500 and 3400 series supports additional model-specific registers listed in Table 2-16. These MSRs also apply to the Intel Core i7 and i5 processor family with a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, or 06_1FH; see Table 2-1.

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Actual maximum turbo frequency is multiplied by 133.33MHz. (Not available in model 06_2EH.)		Package
7:0	Maximum Turbo Ratio Limit 1C (R/O) Maximum Turbo mode ratio limit with 1 core active.	
15:8	Maximum Turbo Ratio Limit 2C (R/O) Maximum Turbo mode ratio limit with 2 cores active.	
23:16	Maximum Turbo Ratio Limit 3C (R/O) Maximum Turbo mode ratio limit with 3 cores active.	
31:24	Maximum Turbo Ratio Limit 4C (R/O) Maximum Turbo mode ratio limit with 4 cores active.	
63:32	Reserved.	
Register Address: 301H, 769	MSR_GQ_SNOOP_MESF	
MSR_GQ_SNOOP_MESF		Package
0	From M to S (R/W)	
1	From E to S (R/W)	
2	From S to S (R/W)	
3	From F to S (R/W)	
4	From M to I (R/W)	

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
5	From E to I (R/W)	
6	From S to I (R/W)	
7	From F to I (R/W)	
63:8	Reserved.	
Register Address: 391H, 913	MSR_UNCORE_PERF_GLOBAL_CTRL	
See Section 22.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 392H, 914	MSR_UNCORE_PERF_GLOBAL_STATUS	
See Section 22.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 393H, 915	MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	
See Section 22.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 394H, 916	MSR_UNCORE_FIXED_CTR0	
See Section 22.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 395H, 917	MSR_UNCORE_FIXED_CTR_CTRL	
See Section 22.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 396H, 918	MSR_UNCORE_ADDR_OPCODE_MATCH	
See Section 22.3.1.2.3, "Uncore Address/Opcode Match MSR."		Package
Register Address: 3B0H, 960	MSR_UNCORE_PMC0	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B1H, 961	MSR_UNCORE_PMC1	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B2H, 962	MSR_UNCORE_PMC2	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B3H, 963	MSR_UNCORE_PMC3	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B4H, 964	MSR_UNCORE_PMC4	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B5H, 965	MSR_UNCORE_PMC5	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B6H, 966	MSR_UNCORE_PMC6	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B7H, 967	MSR_UNCORE_PMC7	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C0H, 944	MSR_UNCORE_PERFEVTSEL0	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C1H, 945	MSR_UNCORE_PERFEVTSEL1	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C2H, 946	MSR_UNCORE_PERFEVTSEL2	

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C3H, 947	MSR_UNCORE_PERFEVTSEL3	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C4H, 948	MSR_UNCORE_PERFEVTSEL4	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C5H, 949	MSR_UNCORE_PERFEVTSEL5	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C6H, 950	MSR_UNCORE_PERFEVTSEL6	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C7H, 951	MSR_UNCORE_PERFEVTSEL7	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package

2.8.2 Additional MSRs in the Intel® Xeon® Processor 7500 Series

The Intel Xeon Processor 7500 series supports MSRs listed in Table 2-15 (except MSR address 1ADH) and additional model-specific registers listed in Table 2-17. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2EH.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Reserved. Attempt to read/write will cause #UD.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 394H, 816	MSR_W_PMON_FIXED_CTR	
Uncore W-box PerfMon fixed counter.		Package
Register Address: 395H, 817	MSR_W_PMON_FIXED_CTR_CTL	
Uncore U-box PerfMon fixed counter control MSR.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 19.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 19.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 19.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 19.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 454H, 1108	IA32_MC21_CTL	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: C00H, 3072	MSR_U_PMON_GLOBAL_CTRL	
Uncore U-box PerfMon global control MSR.		Package
Register Address: C01H, 3073	MSR_U_PMON_GLOBAL_STATUS	
Uncore U-box PerfMon global status MSR.		Package
Register Address: C02H, 3074	MSR_U_PMON_GLOBAL_OVF_CTRL	
Uncore U-box PerfMon global overflow control MSR.		Package
Register Address: C10H, 3088	MSR_U_PMON_EVNT_SEL	
Uncore U-box PerfMon event select MSR.		Package
Register Address: C11H, 3089	MSR_U_PMON_CTR	
Uncore U-box PerfMon counter MSR.		Package
Register Address: C20H, 3104	MSR_B0_PMON_BOX_CTRL	
Uncore B-box 0 PerfMon local box control MSR.		Package
Register Address: C21H, 3105	MSR_B0_PMON_BOX_STATUS	
Uncore B-box 0 PerfMon local box status MSR.		Package
Register Address: C22H, 3106	MSR_B0_PMON_BOX_OVF_CTRL	
Uncore B-box 0 PerfMon local box overflow control MSR.		Package
Register Address: C30H, 3120	MSR_B0_PMON_EVNT_SELO	
Uncore B-box 0 PerfMon event select MSR.		Package
Register Address: C31H, 3121	MSR_B0_PMON_CTR0	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C32H, 3122	MSR_B0_PMON_EVNT_SEL1	
Uncore B-box 0 PerfMon event select MSR.		Package
Register Address: C33H, 3123	MSR_B0_PMON_CTR1	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C34H, 3124	MSR_B0_PMON_EVNT_SEL2	
Uncore B-box 0 PerfMon event select MSR.		Package
Register Address: C35H, 3125	MSR_B0_PMON_CTR2	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C36H, 3126	MSR_B0_PMON_EVNT_SEL3	
Uncore B-box 0 PerfMon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C37H, 3127	MSR_B0_PMON_CTR3	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C40H, 3136	MSR_S0_PMON_BOX_CTRL	
Uncore S-box 0 PerfMon local box control MSR.		Package
Register Address: C41H, 3137	MSR_S0_PMON_BOX_STATUS	
Uncore S-box 0 PerfMon local box status MSR.		Package
Register Address: C42H, 3138	MSR_S0_PMON_BOX_OVF_CTRL	
Uncore S-box 0 PerfMon local box overflow control MSR.		Package
Register Address: C50H, 3152	MSR_S0_PMON_EVNT_SELO	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C51H, 3153	MSR_S0_PMON_CTR0	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C52H, 3154	MSR_S0_PMON_EVNT_SEL1	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C53H, 3155	MSR_S0_PMON_CTR1	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C54H, 3156	MSR_S0_PMON_EVNT_SEL2	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C55H, 3157	MSR_S0_PMON_CTR2	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C56H, 3158	MSR_S0_PMON_EVNT_SEL3	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C57H, 3159	MSR_S0_PMON_CTR3	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C60H, 3168	MSR_B1_PMON_BOX_CTRL	
Uncore B-box 1 PerfMon local box control MSR.		Package
Register Address: C61H, 3169	MSR_B1_PMON_BOX_STATUS	
Uncore B-box 1 PerfMon local box status MSR.		Package
Register Address: C62H, 3170	MSR_B1_PMON_BOX_OVF_CTRL	
Uncore B-box 1 PerfMon local box overflow control MSR.		Package
Register Address: C70H, 3184	MSR_B1_PMON_EVNT_SELO	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C71H, 3185	MSR_B1_PMON_CTR0	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C72H, 3186	MSR_B1_PMON_EVNT_SEL1	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C73H, 3187	MSR_B1_PMON_CTR1	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C74H, 3188	MSR_B1_PMON_EVNT_SEL2	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C75H, 3189	MSR_B1_PMON_CTRL2	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C76H, 3190	MSR_B1_PMON_EVNT_SEL3	
Uncore B-box 1vPerfMon event select MSR.		Package
Register Address: C77H, 3191	MSR_B1_PMON_CTRL3	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C80H, 3120	MSR_W_PMON_BOX_CTRL	
Uncore W-box PerfMon local box control MSR.		Package
Register Address: C81H, 3121	MSR_W_PMON_BOX_STATUS	
Uncore W-box PerfMon local box status MSR.		Package
Register Address: C82H, 3122	MSR_W_PMON_BOX_OVF_CTRL	
Uncore W-box PerfMon local box overflow control MSR.		Package
Register Address: C90H, 3136	MSR_W_PMON_EVNT_SEL0	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C91H, 3137	MSR_W_PMON_CTRL0	
Uncore W-box PerfMon counter MSR.		Package
Register Address: C92H, 3138	MSR_W_PMON_EVNT_SEL1	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C93H, 3139	MSR_W_PMON_CTRL1	
Uncore W-box PerfMon counter MSR.		Package
Register Address: C94H, 3140	MSR_W_PMON_EVNT_SEL2	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C95H, 3141	MSR_W_PMON_CTRL2	
Uncore W-box PerfMon counter MSR.		Package
Register Address: C96H, 3142	MSR_W_PMON_EVNT_SEL3	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C97H, 3143	MSR_W_PMON_CTRL3	
Uncore W-box PerfMon counter MSR.		Package
Register Address: CA0H, 3232	MSR_M0_PMON_BOX_CTRL	
Uncore M-box 0 PerfMon local box control MSR.		Package
Register Address: CA1H, 3233	MSR_M0_PMON_BOX_STATUS	
Uncore M-box 0 PerfMon local box status MSR.		Package
Register Address: CA2H, 3234	MSR_M0_PMON_BOX_OVF_CTRL	
Uncore M-box 0 PerfMon local box overflow control MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CA4H, 3236	MSR_M0_PMON_TIMESTAMP	
Uncore M-box 0 PerfMon time stamp unit select MSR.		Package
Register Address: CA5H, 3237	MSR_M0_PMON_DSP	
Uncore M-box 0 PerfMon DSP unit select MSR.		Package
Register Address: CA6H, 3238	MSR_M0_PMON_ISS	
Uncore M-box 0 PerfMon ISS unit select MSR.		Package
Register Address: CA7H, 3239	MSR_M0_PMON_MAP	
Uncore M-box 0 PerfMon MAP unit select MSR.		Package
Register Address: CA8H, 3240	MSR_M0_PMON_MSC_THR	
Uncore M-box 0 PerfMon MIC THR select MSR.		Package
Register Address: CA9H, 3241	MSR_M0_PMON_PGT	
Uncore M-box 0 PerfMon PGT unit select MSR.		Package
Register Address: CAAH, 3242	MSR_M0_PMON_PLD	
Uncore M-box 0 PerfMon PLD unit select MSR.		Package
Register Address: CABH, 3243	MSR_M0_PMON_ZDP	
Uncore M-box 0 PerfMon ZDP unit select MSR.		Package
Register Address: CBOH, 3248	MSR_M0_PMON_EVNT_SELO	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB1H, 3249	MSR_M0_PMON_CTR0	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB2H, 3250	MSR_M0_PMON_EVNT_SEL1	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB3H, 3251	MSR_M0_PMON_CTR1	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB4H, 3252	MSR_M0_PMON_EVNT_SEL2	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB5H, 3253	MSR_M0_PMON_CTR2	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB6H, 3254	MSR_M0_PMON_EVNT_SEL3	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB7H, 3255	MSR_M0_PMON_CTR3	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB8H, 3256	MSR_M0_PMON_EVNT_SEL4	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB9H, 3257	MSR_M0_PMON_CTR4	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CBAH, 3258	MSR_M0_PMON_EVNT_SEL5	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CBBH, 3259	MSR_M0_PMON_CTR5	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CC0H, 3264	MSR_S1_PMON_BOX_CTRL	
Uncore S-box 1 PerfMon local box control MSR.		Package
Register Address: CC1H, 3265	MSR_S1_PMON_BOX_STATUS	
Uncore S-box 1 PerfMon local box status MSR.		Package
Register Address: CC2H, 3266	MSR_S1_PMON_BOX_OVF_CTRL	
Uncore S-box 1 PerfMon local box overflow control MSR.		Package
Register Address: CD0H, 3280	MSR_S1_PMON_EVNT_SELO	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD1H, 3281	MSR_S1_PMON_CTR0	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CD2H, 3282	MSR_S1_PMON_EVNT_SEL1	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD3H, 3283	MSR_S1_PMON_CTR1	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CD4H, 3284	MSR_S1_PMON_EVNT_SEL2	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD5H, 3285	MSR_S1_PMON_CTR2	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CD6H, 3286	MSR_S1_PMON_EVNT_SEL3	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD7H, 3287	MSR_S1_PMON_CTR3	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CE0H, 3296	MSR_M1_PMON_BOX_CTRL	
Uncore M-box 1 PerfMon local box control MSR.		Package
Register Address: CE1H, 3297	MSR_M1_PMON_BOX_STATUS	
Uncore M-box 1 PerfMon local box status MSR.		Package
Register Address: CE2H, 3298	MSR_M1_PMON_BOX_OVF_CTRL	
Uncore M-box 1 PerfMon local box overflow control MSR.		Package
Register Address: CE4H, 3300	MSR_M1_PMON_TIMESTAMP	
Uncore M-box 1 PerfMon time stamp unit select MSR.		Package
Register Address: CE5H, 3301	MSR_M1_PMON_DSP	
Uncore M-box 1 PerfMon DSP unit select MSR.		Package
Register Address: CE6H, 3302	MSR_M1_PMON_ISS	
Uncore M-box 1 PerfMon ISS unit select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CE7H, 3303	MSR_M1_PMON_MAP	
Uncore M-box 1 PerfMon MAP unit select MSR.		Package
Register Address: CE8H, 3304	MSR_M1_PMON_MSC_THR	
Uncore M-box 1 PerfMon MIC THR select MSR.		Package
Register Address: CE9H, 3305	MSR_M1_PMON_PGT	
Uncore M-box 1 PerfMon PGT unit select MSR.		Package
Register Address: CEAH, 3306	MSR_M1_PMON_PLD	
Uncore M-box 1 PerfMon PLD unit select MSR.		Package
Register Address: CEBH, 3307	MSR_M1_PMON_ZDP	
Uncore M-box 1 PerfMon ZDP unit select MSR.		Package
Register Address: CF0H, 3312	MSR_M1_PMON_EVNT_SELO	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF1H, 3313	MSR_M1_PMON_CTR0	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF2H, 3314	MSR_M1_PMON_EVNT_SEL1	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF3H, 3315	MSR_M1_PMON_CTR1	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF4H, 3316	MSR_M1_PMON_EVNT_SEL2	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF5H, 3317	MSR_M1_PMON_CTR2	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF6H, 3318	MSR_M1_PMON_EVNT_SEL3	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF7H, 3319	MSR_M1_PMON_CTR3	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF8H, 3320	MSR_M1_PMON_EVNT_SEL4	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF9H, 3321	MSR_M1_PMON_CTR4	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CFAH, 3322	MSR_M1_PMON_EVNT_SEL5	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CFBH, 3323	MSR_M1_PMON_CTR5	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: D00H, 3328	MSR_CO_PMON_BOX_CTRL	
Uncore C-box 0 PerfMon local box control MSR.		Package
Register Address: D01H, 3329	MSR_CO_PMON_BOX_STATUS	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 0 PerfMon local box status MSR.		Package
Register Address: D02H, 3330	MSR_CO_PMON_BOX_OVF_CTRL	
Uncore C-box 0 PerfMon local box overflow control MSR.		Package
Register Address: D10H, 3344	MSR_CO_PMON_EVNT_SELO	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D11H, 3345	MSR_CO_PMON_CTR0	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D12H, 3346	MSR_CO_PMON_EVNT_SEL1	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D13H, 3347	MSR_CO_PMON_CTR1	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D14H, 3348	MSR_CO_PMON_EVNT_SEL2	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D15H, 3349	MSR_CO_PMON_CTR2	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D16H, 3350	MSR_CO_PMON_EVNT_SEL3	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D17H, 3351	MSR_CO_PMON_CTR3	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D18H, 3352	MSR_CO_PMON_EVNT_SEL4	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D19H, 3353	MSR_CO_PMON_CTR4	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D1AH, 3354	MSR_CO_PMON_EVNT_SEL5	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D1BH, 3355	MSR_CO_PMON_CTR5	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D20H, 3360	MSR_C4_PMON_BOX_CTRL	
Uncore C-box 4 PerfMon local box control MSR.		Package
Register Address: D21H, 3361	MSR_C4_PMON_BOX_STATUS	
Uncore C-box 4 PerfMon local box status MSR.		Package
Register Address: D22H, 3362	MSR_C4_PMON_BOX_OVF_CTRL	
Uncore C-box 4 PerfMon local box overflow control MSR.		Package
Register Address: D30H, 3376	MSR_C4_PMON_EVNT_SELO	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D31H, 3377	MSR_C4_PMON_CTR0	
Uncore C-box 4 PerfMon counter MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D32H, 3378	MSR_C4_PMON_EVNT_SEL1	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D33H, 3379	MSR_C4_PMON_CTR1	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D34H, 3380	MSR_C4_PMON_EVNT_SEL2	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D35H, 3381	MSR_C4_PMON_CTR2	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D36H, 3382	MSR_C4_PMON_EVNT_SEL3	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D37H, 3383	MSR_C4_PMON_CTR3	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D38H, 3384	MSR_C4_PMON_EVNT_SEL4	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D39H, 3385	MSR_C4_PMON_CTR4	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D3AH, 3386	MSR_C4_PMON_EVNT_SEL5	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D3BH, 3387	MSR_C4_PMON_CTR5	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D40H, 3392	MSR_C2_PMON_BOX_CTRL	
Uncore C-box 2 PerfMon local box control MSR.		Package
Register Address: D41H, 3393	MSR_C2_PMON_BOX_STATUS	
Uncore C-box 2 PerfMon local box status MSR.		Package
Register Address: D42H, 3394	MSR_C2_PMON_BOX_OVF_CTRL	
Uncore C-box 2 PerfMon local box overflow control MSR.		Package
Register Address: D50H, 3408	MSR_C2_PMON_EVNT_SELO	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D51H, 3409	MSR_C2_PMON_CTR0	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D52H, 3410	MSR_C2_PMON_EVNT_SEL1	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D53H, 3411	MSR_C2_PMON_CTR1	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D54H, 3412	MSR_C2_PMON_EVNT_SEL2	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D55H, 3413	MSR_C2_PMON_CTR2	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D56H, 3414	MSR_C2_PMON_EVNT_SEL3	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D57H, 3415	MSR_C2_PMON_CTR3	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D58H, 3416	MSR_C2_PMON_EVNT_SEL4	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D59H, 3417	MSR_C2_PMON_CTR4	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D5AH, 3418	MSR_C2_PMON_EVNT_SEL5	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D5BH, 3419	MSR_C2_PMON_CTR5	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D60H, 3424	MSR_C6_PMON_BOX_CTRL	
Uncore C-box 6 PerfMon local box control MSR.		Package
Register Address: D61H, 3425	MSR_C6_PMON_BOX_STATUS	
Uncore C-box 6 PerfMon local box status MSR.		Package
Register Address: D62H, 3426	MSR_C6_PMON_BOX_OVF_CTRL	
Uncore C-box 6 PerfMon local box overflow control MSR.		Package
Register Address: D70H, 3440	MSR_C6_PMON_EVNT_SELO	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D71H, 3441	MSR_C6_PMON_CTR0	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D72H, 3442	MSR_C6_PMON_EVNT_SEL1	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D73H, 3443	MSR_C6_PMON_CTR1	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D74H, 3444	MSR_C6_PMON_EVNT_SEL2	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D75H, 3445	MSR_C6_PMON_CTR2	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D76H, 3446	MSR_C6_PMON_EVNT_SEL3	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D77H, 3447	MSR_C6_PMON_CTR3	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D78H, 3448	MSR_C6_PMON_EVNT_SEL4	
Uncore C-box 6 PerfMon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D79H, 3449	MSR_C6_PMON_CTR4	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D7AH, 3450	MSR_C6_PMON_EVNT_SEL5	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D7BH, 3451	MSR_C6_PMON_CTR5	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D80H, 3456	MSR_C1_PMON_BOX_CTRL	
Uncore C-box 1 PerfMon local box control MSR.		Package
Register Address: D81H, 3457	MSR_C1_PMON_BOX_STATUS	
Uncore C-box 1 PerfMon local box status MSR.		Package
Register Address: D82H, 3458	MSR_C1_PMON_BOX_OVF_CTRL	
Uncore C-box 1 PerfMon local box overflow control MSR.		Package
Register Address: D90H, 3472	MSR_C1_PMON_EVNT_SELO	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D91H, 3473	MSR_C1_PMON_CTR0	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D92H, 3474	MSR_C1_PMON_EVNT_SEL1	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D93H, 3475	MSR_C1_PMON_CTR1	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D94H, 3476	MSR_C1_PMON_EVNT_SEL2	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D95H, 3477	MSR_C1_PMON_CTR2	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D96H, 3478	MSR_C1_PMON_EVNT_SEL3	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D97H, 3479	MSR_C1_PMON_CTR3	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D98H, 3480	MSR_C1_PMON_EVNT_SEL4	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D99H, 3481	MSR_C1_PMON_CTR4	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D9AH, 3482	MSR_C1_PMON_EVNT_SEL5	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D9BH, 3483	MSR_C1_PMON_CTR5	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: DA0H, 3488	MSR_C5_PMON_BOX_CTRL	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 5 PerfMon local box control MSR.		Package
Register Address: DA1H, 3489	MSR_C5_PMON_BOX_STATUS	
Uncore C-box 5 PerfMon local box status MSR.		Package
Register Address: DA2H, 3490	MSR_C5_PMON_BOX_OVF_CTRL	
Uncore C-box 5 PerfMon local box overflow control MSR.		Package
Register Address: DB0H, 3504	MSR_C5_PMON_EVNT_SELO	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB1H, 3505	MSR_C5_PMON_CTR0	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB2H, 3506	MSR_C5_PMON_EVNT_SEL1	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB3H, 3507	MSR_C5_PMON_CTR1	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB4H, 3508	MSR_C5_PMON_EVNT_SEL2	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB5H, 3509	MSR_C5_PMON_CTR2	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB6H, 3510	MSR_C5_PMON_EVNT_SEL3	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB7H, 3511	MSR_C5_PMON_CTR3	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB8H, 3512	MSR_C5_PMON_EVNT_SEL4	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB9H, 3513	MSR_C5_PMON_CTR4	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DBAH, 3514	MSR_C5_PMON_EVNT_SEL5	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DBBH, 3515	MSR_C5_PMON_CTR5	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DC0H, 3520	MSR_C3_PMON_BOX_CTRL	
Uncore C-box 3 PerfMon local box control MSR.		Package
Register Address: DC1H, 3521	MSR_C3_PMON_BOX_STATUS	
Uncore C-box 3 PerfMon local box status MSR.		Package
Register Address: DC2H, 3522	MSR_C3_PMON_BOX_OVF_CTRL	
Uncore C-box 3 PerfMon local box overflow control MSR.		Package
Register Address: DDOH, 3536	MSR_C3_PMON_EVNT_SELO	
Uncore C-box 3 PerfMon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DD1H, 3537	MSR_C3_PMON_CTRL0	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD2H, 3538	MSR_C3_PMON_EVNT_SEL1	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD3H, 3539	MSR_C3_PMON_CTRL1	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD4H, 3540	MSR_C3_PMON_EVNT_SEL2	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD5H, 3541	MSR_C3_PMON_CTRL2	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD6H, 3542	MSR_C3_PMON_EVNT_SEL3	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD7H, 3543	MSR_C3_PMON_CTRL3	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD8H, 3544	MSR_C3_PMON_EVNT_SEL4	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD9H, 3545	MSR_C3_PMON_CTRL4	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DDAH, 3546	MSR_C3_PMON_EVNT_SEL5	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DDBH, 3547	MSR_C3_PMON_CTRL5	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DE0H, 3552	MSR_C7_PMON_BOX_CTRL	
Uncore C-box 7 PerfMon local box control MSR.		Package
Register Address: DE1H, 3553	MSR_C7_PMON_BOX_STATUS	
Uncore C-box 7 PerfMon local box status MSR.		Package
Register Address: DE2H, 3554	MSR_C7_PMON_BOX_OVF_CTRL	
Uncore C-box 7 PerfMon local box overflow control MSR.		Package
Register Address: DF0H, 3568	MSR_C7_PMON_EVNT_SELO	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF1H, 3569	MSR_C7_PMON_CTRL0	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF2H, 3570	MSR_C7_PMON_EVNT_SEL1	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF3H, 3571	MSR_C7_PMON_CTRL1	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF4H, 3572	MSR_C7_PMON_EVNT_SEL2	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF5H, 3573	MSR_C7_PMON_CTR2	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF6H, 3574	MSR_C7_PMON_EVNT_SEL3	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF7H, 3575	MSR_C7_PMON_CTR3	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF8H, 3576	MSR_C7_PMON_EVNT_SEL4	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF9H, 3577	MSR_C7_PMON_CTR4	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DFAH, 3578	MSR_C7_PMON_EVNT_SEL5	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DFBH, 3579	MSR_C7_PMON_CTR5	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: E00H, 3584	MSR_R0_PMON_BOX_CTRL	
Uncore R-box 0 PerfMon local box control MSR.		Package
Register Address: E01H, 3585	MSR_R0_PMON_BOX_STATUS	
Uncore R-box 0 PerfMon local box status MSR.		Package
Register Address: E02H, 3586	MSR_R0_PMON_BOX_OVF_CTRL	
Uncore R-box 0 PerfMon local box overflow control MSR.		Package
Register Address: E04H, 3588	MSR_R0_PMON_IPERF0_P0	
Uncore R-box 0 PerfMon IPERF0 unit Port 0 select MSR.		Package
Register Address: E05H, 3589	MSR_R0_PMON_IPERF0_P1	
Uncore R-box 0 PerfMon IPERF0 unit Port 1 select MSR.		Package
Register Address: E06H, 3590	MSR_R0_PMON_IPERF0_P2	
Uncore R-box 0 PerfMon IPERF0 unit Port 2 select MSR.		Package
Register Address: E07H, 3591	MSR_R0_PMON_IPERF0_P3	
Uncore R-box 0 PerfMon IPERF0 unit Port 3 select MSR.		Package
Register Address: E08H, 3592	MSR_R0_PMON_IPERF0_P4	
Uncore R-box 0 PerfMon IPERF0 unit Port 4 select MSR.		Package
Register Address: E09H, 3593	MSR_R0_PMON_IPERF0_P5	
Uncore R-box 0 PerfMon IPERF0 unit Port 5 select MSR.		Package
Register Address: E0AH, 3594	MSR_R0_PMON_IPERF0_P6	
Uncore R-box 0 PerfMon IPERF0 unit Port 6 select MSR.		Package
Register Address: E0BH, 3595	MSR_R0_PMON_IPERF0_P7	
Uncore R-box 0 PerfMon IPERF0 unit Port 7 select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E0CH, 3596	MSR_R0_PMON_QLX_P0	
Uncore R-box 0 PerfMon QLX unit Port 0 select MSR.		Package
Register Address: E0DH, 3597	MSR_R0_PMON_QLX_P1	
Uncore R-box 0 PerfMon QLX unit Port 1 select MSR.		Package
Register Address: E0EH, 3598	MSR_R0_PMON_QLX_P2	
Uncore R-box 0 PerfMon QLX unit Port 2 select MSR.		Package
Register Address: E0FH, 3599	MSR_R0_PMON_QLX_P3	
Uncore R-box 0 PerfMon QLX unit Port 3 select MSR.		Package
Register Address: E10H, 3600	MSR_R0_PMON_EVNT_SELO	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E11H, 3601	MSR_R0_PMON_CTR0	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E12H, 3602	MSR_R0_PMON_EVNT_SEL1	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E13H, 3603	MSR_R0_PMON_CTR1	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E14H, 3604	MSR_R0_PMON_EVNT_SEL2	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E15H, 3605	MSR_R0_PMON_CTR2	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E16H, 3606	MSR_R0_PMON_EVNT_SEL3	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E17H, 3607	MSR_R0_PMON_CTR3	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E18H, 3608	MSR_R0_PMON_EVNT_SEL4	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E19H, 3609	MSR_R0_PMON_CTR4	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E1AH, 3610	MSR_R0_PMON_EVNT_SEL5	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E1BH, 3611	MSR_R0_PMON_CTR5	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E1CH, 3612	MSR_R0_PMON_EVNT_SEL6	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E1DH, 3613	MSR_R0_PMON_CTR6	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E1EH, 3614	MSR_R0_PMON_EVNT_SEL7	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E1FH, 3615	MSR_R0_PMON_CTR7	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E20H, 3616	MSR_R1_PMON_BOX_CTRL	
Uncore R-box 1 PerfMon local box control MSR.		Package
Register Address: E21H, 3617	MSR_R1_PMON_BOX_STATUS	
Uncore R-box 1 PerfMon local box status MSR.		Package
Register Address: E22H, 3618	MSR_R1_PMON_BOX_OVF_CTRL	
Uncore R-box 1 PerfMon local box overflow control MSR.		Package
Register Address: E24H, 3620	MSR_R1_PMON_IPERF1_P8	
Uncore R-box 1 PerfMon IPERF1 unit Port 8 select MSR.		Package
Register Address: E25H, 3621	MSR_R1_PMON_IPERF1_P9	
Uncore R-box 1 PerfMon IPERF1 unit Port 9 select MSR.		Package
Register Address: E26H, 3622	MSR_R1_PMON_IPERF1_P10	
Uncore R-box 1 PerfMon IPERF1 unit Port 10 select MSR.		Package
Register Address: E27H, 3623	MSR_R1_PMON_IPERF1_P11	
Uncore R-box 1 PerfMon IPERF1 unit Port 11 select MSR.		Package
Register Address: E28H, 3624	MSR_R1_PMON_IPERF1_P12	
Uncore R-box 1 PerfMon IPERF1 unit Port 12 select MSR.		Package
Register Address: E29H, 3625	MSR_R1_PMON_IPERF1_P13	
Uncore R-box 1 PerfMon IPERF1 unit Port 13 select MSR.		Package
Register Address: E2AH, 3626	MSR_R1_PMON_IPERF1_P14	
Uncore R-box 1 PerfMon IPERF1 unit Port 14 select MSR.		Package
Register Address: E2BH, 3627	MSR_R1_PMON_IPERF1_P15	
Uncore R-box 1 PerfMon IPERF1 unit Port 15 select MSR.		Package
Register Address: E2CH, 3628	MSR_R1_PMON_QLX_P4	
Uncore R-box 1 PerfMon QLX unit Port 4 select MSR.		Package
Register Address: E2DH, 3629	MSR_R1_PMON_QLX_P5	
Uncore R-box 1 PerfMon QLX unit Port 5 select MSR.		Package
Register Address: E2EH, 3630	MSR_R1_PMON_QLX_P6	
Uncore R-box 1 PerfMon QLX unit Port 6 select MSR.		Package
Register Address: E2FH, 3631	MSR_R1_PMON_QLX_P7	
Uncore R-box 1 PerfMon QLX unit Port 7 select MSR.		Package
Register Address: E30H, 3632	MSR_R1_PMON_EVNT_SEL8	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E31H, 3633	MSR_R1_PMON_CTR8	
Uncore R-box 1 PerfMon counter MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E32H, 3634	MSR_R1_PMON_EVNT_SEL9	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E33H, 3635	MSR_R1_PMON_CTR9	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E34H, 3636	MSR_R1_PMON_EVNT_SEL10	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E35H, 3637	MSR_R1_PMON_CTR10	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E36H, 3638	MSR_R1_PMON_EVNT_SEL11	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E37H, 3639	MSR_R1_PMON_CTR11	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E38H, 3640	MSR_R1_PMON_EVNT_SEL12	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E39H, 3641	MSR_R1_PMON_CTR12	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E3AH, 3642	MSR_R1_PMON_EVNT_SEL13	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E3BH, 3643	MSR_R1_PMON_CTR13	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E3CH, 3644	MSR_R1_PMON_EVNT_SEL14	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E3DH, 3645	MSR_R1_PMON_CTR14	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E3EH, 3646	MSR_R1_PMON_EVNT_SEL15	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E3FH, 3647	MSR_R1_PMON_CTR15	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E45H, 3653	MSR_B0_PMON_MATCH	
Uncore B-box 0 PerfMon local box match MSR.		Package
Register Address: E46H, 3654	MSR_B0_PMON_MASK	
Uncore B-box 0 PerfMon local box mask MSR.		Package
Register Address: E49H, 3657	MSR_S0_PMON_MATCH	
Uncore S-box 0 PerfMon local box match MSR.		Package
Register Address: E4AH, 3658	MSR_S0_PMON_MASK	
Uncore S-box 0 PerfMon local box mask MSR.		Package
Register Address: E4DH, 3661	MSR_B1_PMON_MATCH	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore B-box 1 PerfMon local box match MSR.		Package
Register Address: E4EH, 3662	MSR_B1_PMON_MASK	
Uncore B-box 1 PerfMon local box mask MSR.		Package
Register Address: E54H, 3668	MSR_M0_PMON_MM_CONFIG	
Uncore M-box 0 PerfMon local box address match/mask config MSR.		Package
Register Address: E55H, 3669	MSR_M0_PMON_ADDR_MATCH	
Uncore M-box 0 PerfMon local box address match MSR.		Package
Register Address: E56H, 3670	MSR_M0_PMON_ADDR_MASK	
Uncore M-box 0 PerfMon local box address mask MSR.		Package
Register Address: E59H, 3673	MSR_S1_PMON_MATCH	
Uncore S-box 1 PerfMon local box match MSR.		Package
Register Address: E5AH, 3674	MSR_S1_PMON_MASK	
Uncore S-box 1 PerfMon local box mask MSR.		Package
Register Address: E5CH, 3676	MSR_M1_PMON_MM_CONFIG	
Uncore M-box 1 PerfMon local box address match/mask config MSR.		Package
Register Address: E5DH, 3677	MSR_M1_PMON_ADDR_MATCH	
Uncore M-box 1 PerfMon local box address match MSR.		Package
Register Address: E5EH, 3678	MSR_M1_PMON_ADDR_MASK	
Uncore M-box 1 PerfMon local box address mask MSR.		Package
Register Address: 3B5H, 965	MSR_UNCORE_PMC5	
See Section 22.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package

2.9 MSRS IN THE INTEL® XEON® PROCESSOR 5600 SERIES BASED ON WESTMERE MICROARCHITECTURE

The Intel® Xeon® Processor 5600 Series is based on Westmere microarchitecture and supports the MSR interfaces listed in Table 2-15, Table 2-16, plus additional MSRs listed in Table 2-18. These MSRs apply to the Intel Core i7, i5, and i3 processor family with a CPUID Signature DisplayFamily_DisplayModel value of 06_25H or 06_2CH; see Table 2-1.

Table 2-18. Additional MSRs Supported by Intel® Processors Based on Westmere Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core

Table 2-18. Additional MSRs Supported by Intel® Processors Based on Westmere Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
63:48	Reserved.	
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package

2.10 MSRS IN THE INTEL® XEON® PROCESSOR E7 FAMILY BASED ON WESTMERE MICROARCHITECTURE

The Intel® Xeon® Processor E7 Family is based on the Westmere microarchitecture and supports the MSR interfaces listed in Table 2-15 (except MSR address 1ADH), Table 2-16, plus additional MSRs listed in Table 2-19. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2FH.

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Reserved. Attempt to read/write will cause #UD.		Package
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package
Register Address: F40H, 3904	MSR_C8_PMON_BOX_CTRL	
Uncore C-box 8 PerfMon local box control MSR.		Package
Register Address: F41H, 3905	MSR_C8_PMON_BOX_STATUS	
Uncore C-box 8 PerfMon local box status MSR.		Package
Register Address: F42H, 3906	MSR_C8_PMON_BOX_OVF_CTRL	
Uncore C-box 8 PerfMon local box overflow control MSR.		Package
Register Address: F50H, 3920	MSR_C8_PMON_EVNT_SELO	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F51H, 3921	MSR_C8_PMON_CTR0	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F52H, 3922	MSR_C8_PMON_EVNT_SEL1	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F53H, 3923	MSR_C8_PMON_CTR1	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F54H, 3924	MSR_C8_PMON_EVNT_SEL2	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F55H, 3925	MSR_C8_PMON_CTR2	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F56H, 3926	MSR_C8_PMON_EVNT_SEL3	
Uncore C-box 8 PerfMon event select MSR.		Package

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: F57H, 3927	MSR_C8_PMON_CTR3	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F58H, 3928	MSR_C8_PMON_EVNT_SEL4	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F59H, 3929	MSR_C8_PMON_CTR4	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F5AH, 3930	MSR_C8_PMON_EVNT_SEL5	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F5BH, 3931	MSR_C8_PMON_CTR5	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: FC0H, 4032	MSR_C9_PMON_BOX_CTRL	
Uncore C-box 9 PerfMon local box control MSR.		Package
Register Address: FC1H, 4033	MSR_C9_PMON_BOX_STATUS	
Uncore C-box 9 PerfMon local box status MSR.		Package
Register Address: FC2H, 4034	MSR_C9_PMON_BOX_OVF_CTRL	
Uncore C-box 9 PerfMon local box overflow control MSR.		Package
Register Address: FDOH, 4048	MSR_C9_PMON_EVNT_SELO	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD1H, 4049	MSR_C9_PMON_CTR0	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD2H, 4050	MSR_C9_PMON_EVNT_SEL1	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD3H, 4051	MSR_C9_PMON_CTR1	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD4H, 4052	MSR_C9_PMON_EVNT_SEL2	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD5H, 4053	MSR_C9_PMON_CTR2	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD6H, 4054	MSR_C9_PMON_EVNT_SEL3	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD7H, 4055	MSR_C9_PMON_CTR3	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD8H, 4056	MSR_C9_PMON_EVNT_SEL4	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD9H, 4057	MSR_C9_PMON_CTR4	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FDAH, 4058	MSR_C9_PMON_EVNT_SEL5	

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FDBH, 4059	MSR_C9_PMON_CTR5	
Uncore C-box 9 PerfMon counter MSR.		Package

2.11 MSRS IN THE INTEL® PROCESSOR FAMILY BASED ON SANDY BRIDGE MICROARCHITECTURE

Table 2-20 lists model-specific registers (MSRs) that are common to the Intel® processor family based on Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH or 06_2DH; see Table 2-1. Additional MSRs specific to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH are listed in Table 2-21.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Thread
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Thread
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, "Time-Stamp Counter," and see Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O) Count SMIs.	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	
2	Enable VMX Outside SMX Operation (R/WL)	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Thread
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Thread
Register Address: C5H, 197	IA32_PMC4	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C6H, 198	IA32_PMC5	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C7H, 199	IA32_PMC6	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C8H, 200	IA32_PMC7	
Performance Counter Register (if core not shared by threads)		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	
26	C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Enable C3 Undemotion (R/W) When set, enables undemotion from demoted C3.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
28	Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.	
63:29	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Core
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	C-State Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit 10]: 000b - C3 is the max C-State to include. 001b - C6 is the max C-State to include. 010b - C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Thread
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Thread
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFVTSEL0	
See Table 2-2.		Thread
Register Address: 187H, 391	IA32_PERFVTSEL1	
See Table 2-2.		Thread
Register Address: 188H, 392	IA32_PERFVTSEL2	
See Table 2-2.		Thread
Register Address: 189H, 393	IA32_PERFVTSEL3	
See Table 2-2.		Thread
Register Address: 18AH, 394	IA32_PERFVTSEL4	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 4.		Core
Register Address: 18BH, 395	IA32_PERFVTSEL5	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 5.		Core
Register Address: 18CH, 396	IA32_PERFVTSEL6	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 6.		Core
Register Address: 18DH, 397	IA32_PERFVTSEL7	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 7.		Core
Register Address: 198H, 408	IA32_PERF_STATUS	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Package
15:0	Current Performance State Value	
63:16	Reserved.	
Register Address: 198H, 408	MSR_PERF_STATUS	
Performance Status		Package
47:32	Core Voltage (R/O) P-state core voltage can be computed by MSR_PERF_STATUS[37:32] * (float) 1/(2 ¹³).	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Thread
3:0	On demand Clock Modulation Duty Cycle (R/W) In 6.25% increment.	
4	On demand Clock Modulation Enable (R/W)	
63:5	Reserved.	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
7	Thermal Threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WC0) See Table 2-2.	
15:12	Reserved.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Thread
6:1	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Thread
10:8	Reserved	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Thread
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Thread
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Thread
21:19	Reserved.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	Limit CPUID Maxval (R/W) See Table 2-2.	Thread
23	xTPR Message Disable (R/W) See Table 2-2.	Thread
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Thread
37:35	Reserved.	
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Package
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Unique
15:0	Reserved.	
23:16	Temperature Target (R) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
63:24	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	L2 Adjacent Cache Line Prefetcher Disable (R/W) If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	Core
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
3	DCU IP Prefetcher Disable (R/W) If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	Core
63:4	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org .		
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package
Register Address: 1B1H, 433	IA32_PACKAGE_THERM_STATUS	
See Table 2-2.		Package
Register Address: 1B2H, 434	IA32_PACKAGE_THERM_INTERRUPT	
See Table 2-2.		Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 20.9.2, "Filtering of Last Branch Records."		Thread
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
0	LBR: Last Branch Record	
1	BTF	
5:2	Reserved.	
6	TR: Branch Trace	
7	BTS: Log Branch Trace Message to BTS buffer	
8	BTINT	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
9	BTS_OFF_OS	
10	BTS_OFF_USER	
11	FREEZE_LBR_ON_PMI	
12	FREEZE_PERFMON_ON_PMI	
13	ENABLE_UNCORE_PMI	
14	FREEZE_WHILE_SMM	
63:15	Reserved.	
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
See http://biosbits.org .		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Thread
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Thread
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Thread
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Thread
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Thread
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Thread
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Thread
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Thread
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Thread
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Thread
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Thread
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Thread
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Thread
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Thread
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Thread
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Thread
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Thread
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Thread
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
See Table 2-2.		Thread
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
See Table 2-2.		Thread
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Thread
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Thread
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Thread
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Thread
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Thread
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Thread
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Thread
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Thread
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Thread
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Thread
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Thread
Register Address: 280H, 640	IA32_MCO_CTL2	
See Table 2-2.		Core
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Core
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
Always 0 (CMCI not supported).		Package
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Thread
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2 and Section 20.4.1, "IA32_DEBUGCTL MSR."		Thread
5:0	LBR Format See Table 2-2.	
6	PEBS Record Format.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
7	PEBSSaveArchRegs See Table 2-2.	
11:8	PEBS_REC_FORMAT See Table 2-2.	
12	SMM_FREEZE See Table 2-2.	
63:13	Reserved.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 22.6.2.2, "Global Counter Control Facilities."		
0	Ovf_PMC0	Thread
1	Ovf_PMC1	Thread
2	Ovf_PMC2	Thread
3	Ovf_PMC3	Thread
4	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)	Core
5	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)	Core
6	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)	Core
7	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)	Core
31:8	Reserved.	
32	Ovf_FixedCtr0	Thread
33	Ovf_FixedCtr1	Thread
34	Ovf_FixedCtr2	Thread
60:35	Reserved.	
61	Ovf_Uncore	Thread
62	Ovf_BufDSSAVE	Thread
63	CondChgd	Thread
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2 and Section 22.6.2.2, "Global Counter Control Facilities."		Thread
0	Set 1 to enable PMC0 to count.	Thread
1	Set 1 to enable PMC1 to count.	Thread
2	Set 1 to enable PMC2 to count.	Thread
3	Set 1 to enable PMC3 to count.	Thread
4	Set 1 to enable PMC4 to count (if CPUID.0AH:EAX[15:8] > 4).	Core
5	Set 1 to enable PMC5 to count (if CPUID.0AH:EAX[15:8] > 5).	Core
6	Set 1 to enable PMC6 to count (if CPUID.0AH:EAX[15:8] > 6).	Core
7	Set 1 to enable PMC7 to count (if CPUID.0AH:EAX[15:8] > 7).	Core

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
31:8	Reserved.	
32	Set 1 to enable FixedCtr0 to count.	Thread
33	Set 1 to enable FixedCtr1 to count.	Thread
34	Set 1 to enable FixedCtr2 to count.	Thread
63:35	Reserved.	
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2 and Section 22.6.2.2, "Global Counter Control Facilities."		
0	Set 1 to clear Ovf_PMC0.	Thread
1	Set 1 to clear Ovf_PMC1.	Thread
2	Set 1 to clear Ovf_PMC2.	Thread
3	Set 1 to clear Ovf_PMC3.	Thread
4	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).	Core
5	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).	Core
6	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).	Core
7	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).	Core
31:8	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	Thread
33	Set 1 to clear Ovf_FixedCtr1.	Thread
34	Set 1 to clear Ovf_FixedCtr2.	Thread
60:35	Reserved.	
61	Set 1 to clear Ovf_Uncore.	Thread
62	Set 1 to clear Ovf_BufDSSAVE.	Thread
63	Set 1 to clear CondChgd.	Thread
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 22.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
0	Enable PEBS on IA32_PMC0. (R/W)	
1	Enable PEBS on IA32_PMC1. (R/W)	
2	Enable PEBS on IA32_PMC2. (R/W)	
3	Enable PEBS on IA32_PMC3. (R/W)	
31:4	Reserved.	
32	Enable Load Latency on IA32_PMC0. (R/W)	
33	Enable Load Latency on IA32_PMC1. (R/W)	
34	Enable Load Latency on IA32_PMC2. (R/W)	
35	Enable Load Latency on IA32_PMC3. (R/W)	
62:36	Reserved.	
63	Enable Precise Store (R/W)	
Register Address: 3F6H, 1014	MSR_PEBS_LD_LAT	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 22.3.1.1.2, "Load Latency Performance Monitoring Facility."		Thread
15:0	Minimum threshold latency value of tagged load operation that will be counted. (R/W)	
63:36	Reserved.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C7 Residency Counter (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FEH, 1022	MSR_CORE_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C7 Residency Counter (R/O) Value since last reset that this core is in processor-specific C7 states. Count at the same frequency as the TSC.	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Core

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Core
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 403H, 1027	IA32_MCO_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Core
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 407H, 1031	IA32_MC1_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
0	PCU Hardware Error (R/W) When set, enables signaling of PCU hardware detected errors.	
1	PCU Controller Error (R/W) When set, enables signaling of PCU controller detected errors.	
2	PCU Firmware Error (R/W) When set, enables signaling of PCU firmware detected errors.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:2	Reserved.	
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Core
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		Thread
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2 and Appendix A.4, "VM-Exit Controls."		Thread
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2 and Appendix A.5, "VM-Entry Controls."		Thread
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2 and Appendix A.6, "Miscellaneous Data."		Thread
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2 and Appendix A.9, "VMCS Enumeration."		Thread
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2		Thread
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTLs	
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2		Thread
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTLs	
Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2		Thread
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTLs	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2		Thread
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTLs	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2		Thread
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Thread
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Thread
Register Address: 4C3H, 1219	IA32_A_PMC2	
See Table 2-2.		Thread
Register Address: 4C4H, 1220	IA32_A_PMC3	
See Table 2-2.		Thread
Register Address: 4C5H, 1221	IA32_A_PMC4	
See Table 2-2.		Core
Register Address: 4C6H, 1222	IA32_A_PMC5	
See Table 2-2.		Core
Register Address: 4C7H, 1223	IA32_A_PMC6	
See Table 2-2.		Core
Register Address: 4C8H, 1224	IA32_A_PMC7	
See Table 2-2.		Core
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 22.6.3.4, "Debug Store (DS) Mechanism."		Thread
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 17.10.1, "RAPL Interfaces."		Package
Register Address: 60AH, 1546	MSR_PKG_C3_IRT_L	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C3 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60BH, 1547	MSR_PKG_C6_IRTL	
Package C6 Interrupt Response Limit (R/W) This MSR defines the budget allocated for the package to exit from a C6 to a C0 state, where an interrupt request can be delivered to the core and serviced. Additional core-exit latency may be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:16	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C2 Residency Counter (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 638H, 1592	MSR_PP0_POWER_LIMIT	
PP0 RAPL Power Limit Control (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.9.1 and record format in Section 20.4.8.1.		Thread
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.		Thread
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
See Table 2-2.		Thread
Register Address: 802H–83FH, 2050–2111	X2APIC MSRs	
See Table 2-2.		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 20.17.2, “IA32_TSC_AUX Register and RDTSCP Support.”		Thread

2.11.1 MSRs in the 2nd Generation Intel® Core™ Processor Family Based on Sandy Bridge Microarchitecture

Table 2-21 and Table 2-22 list model-specific registers (MSRs) that are specific to the 2nd generation Intel® Core™ processor family based on the Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH; see Table 2-1.

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 60CH, 1548	MSR_PKG_C7_IRTL	
Package C7 Interrupt Response Limit (R/W) This MSR defines the budget allocated for the package to exit from a C7 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency may be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 63AH, 1594	MSR_PPO_POLICY	

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
PP0 Balance Policy (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 640H, 1600	MSR_PP1_POWER_LIMIT	
PP1 RAPL Power Limit Control (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	
PP1 Energy Status (R/O) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 642H, 1602	MSR_PP1_POLICY	
PP1 Balance Policy (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
See Table 2-20, Table 2-21, and Table 2-22 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH.		

Table 2-22 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH.

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 391H, 913	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	
3	Slice 3 select.	
4	Slice 4 select.	
18:5	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: 392H, 914	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Report the number of C-Box units with performance counters, including processor cores and processor graphics.	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSELO	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb unit, Counter 1 Event Select MSR		Package
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 702H, 1794	MSR_UNC_CBO_0_PERFEVTSEL2	
Uncore C-Box 0, Counter 2 Event Select MSR		Package
Register Address: 703H, 1795	MSR_UNC_CBO_0_PERFEVTSEL3	
Uncore C-Box 0, Counter 3 Event Select MSR		Package
Register Address: 705H, 1797	MSR_UNC_CBO_0_UNIT_STATUS	
Uncore C-Box 0, Unit Status for Counter 0-3		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 708H, 1800	MSR_UNC_CBO_0_PERFCTR2	
Uncore C-Box 0, Performance Counter 2		Package
Register Address: 709H, 1801	MSR_UNC_CBO_0_PERFCTR3	
Uncore C-Box 0, Performance Counter 3		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 712H, 1810	MSR_UNC_CBO_1_PERFEVTSEL2	
Uncore C-Box 1, Counter 2 Event Select MSR		Package
Register Address: 713H, 1811	MSR_UNC_CBO_1_PERFEVTSEL3	
Uncore C-Box 1, Counter 3 Event Select MSR		Package
Register Address: 715H, 1813	MSR_UNC_CBO_1_UNIT_STATUS	
Uncore C-Box 1, Unit Status for Counter 0-3		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 718H, 1816	MSR_UNC_CBO_1_PERFCTR2	
Uncore C-Box 1, Performance Counter 2		Package
Register Address: 719H, 1817	MSR_UNC_CBO_1_PERFCTR3	
Uncore C-Box 1, Performance Counter 3		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 722H, 1826	MSR_UNC_CBO_2_PERFEVTSEL2	
Uncore C-Box 2, Counter 2 Event Select MSR		Package
Register Address: 723H, 1827	MSR_UNC_CBO_2_PERFEVTSEL3	
Uncore C-Box 2, Counter 3 Event Select MSR		Package
Register Address: 725H, 1829	MSR_UNC_CBO_2_UNIT_STATUS	
Uncore C-Box 2, Unit Status for Counter 0-3		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 728H, 1832	MSR_UNC_CBO_3_PERFCTR2	

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 3, Performance Counter 2		Package
Register Address: 729H, 1833	MSR_UNC_CBO_3_PERFCTR3	
Uncore C-Box 3, Performance Counter 3		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 732H, 1842	MSR_UNC_CBO_3_PERFEVTSEL2	
Uncore C-Box 3, Counter 2 Event Select MSR		Package
Register Address: 733H, 1843	MSR_UNC_CBO_3_PERFEVTSEL3	
Uncore C-Box 3, counter 3 Event Select MSR		Package
Register Address: 735H, 1845	MSR_UNC_CBO_3_UNIT_STATUS	
Uncore C-Box 3, Unit Status for Counter 0-3		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: 738H, 1848	MSR_UNC_CBO_3_PERFCTR2	
Uncore C-Box 3, Performance Counter 2		Package
Register Address: 739H, 1849	MSR_UNC_CBO_3_PERFCTR3	
Uncore C-Box 3, Performance Counter 3		Package
Register Address: 740H, 1856	MSR_UNC_CBO_4_PERFEVTSELO	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 741H, 1857	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package
Register Address: 742H, 1858	MSR_UNC_CBO_4_PERFEVTSEL2	
Uncore C-Box 4, Counter 2 Event Select MSR		Package
Register Address: 743H, 1859	MSR_UNC_CBO_4_PERFEVTSEL3	
Uncore C-Box 4, Counter 3 Event Select MSR		Package
Register Address: 745H, 1861	MSR_UNC_CBO_4_UNIT_STATUS	
Uncore C-Box 4, Unit status for Counter 0-3		Package
Register Address: 746H, 1862	MSR_UNC_CBO_4_PERFCTR0	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 747H, 1863	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 748H, 1864	MSR_UNC_CBO_4_PERFCTR2	
Uncore C-Box 4, Performance Counter 2		Package

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 749H, 1865	MSR_UNC_CBO_4_PERFCTR3	
Uncore C-Box 4, Performance Counter 3		Package

2.11.2 MSRs in the Intel® Xeon® Processor E5 Family Based on Sandy Bridge Microarchitecture

Table 2-23 lists additional model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 Family based on Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH, and also support MSRs listed in Table 2-20 and Table 2-24.

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17FH, 383	MSR_ERROR_CONTROL	
	MC Bank Error Configuration (R/W)	Package
0	Reserved.	
1	MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.	
63:2	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 cores active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 cores active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 cores active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 cores active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 cores active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 cores active.	Package
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 cores active.	Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 39CH, 924	MSR_PEBS_NUM_ALT	
ENABLE_PEBS_NUM_ALT (R/W)		Package
0	ENABLE_PEBS_NUM_ALT (R/W) Write 1 to enable alternate PEBS counting logic for specific events requiring additional configuration, see https://perfmon-events.intel.com/ .	
63:1	Reserved, must be zero.	
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 19.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 19.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 19.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 19.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 19.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 19.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
Package RAPL Perf Status (R/O)		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
See Table 2-20, Table 2-23, and Table 2-24 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH.		

2.11.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 Family

Intel Xeon Processor E5 family is based on the Sandy Bridge microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-24. For complete details of the uncore PMU, refer to the Intel Xeon Processor E5 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH.

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C08H, 3080	MSR_U_PMON_UCLK_FIXED_CTL	
Uncore U-box UCLK Fixed Counter Control		Package
Register Address: C09H, 3081	MSR_U_PMON_UCLK_FIXED_CTR	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore U-box UCLK Fixed Counter		Package
Register Address: C10H, 3088	MSR_U_PMON_EVTNSELO	
Uncore U-box PerfMon Event Select for U-box Counter 0		Package
Register Address: C11H, 3089	MSR_U_PMON_EVTNSEL1	
Uncore U-box PerfMon Event Select for U-box Counter 1		Package
Register Address: C16H, 3094	MSR_U_PMON_CTR0	
Uncore U-box PerfMon Counter 0		Package
Register Address: C17H, 3095	MSR_U_PMON_CTR1	
Uncore U-box PerfMon Counter 1		Package
Register Address: C24H, 3108	MSR_PCU_PMON_BOX_CTL	
Uncore PCU PerfMon for PCU-box-wide Control		Package
Register Address: C30H, 3120	MSR_PCU_PMON_EVTNSELO	
Uncore PCU PerfMon Event Select for PCU Counter 0		Package
Register Address: C31H, 3121	MSR_PCU_PMON_EVTNSEL1	
Uncore PCU PerfMon Event Select for PCU Counter 1		Package
Register Address: C32H, 3122	MSR_PCU_PMON_EVTNSEL2	
Uncore PCU PerfMon Event Select for PCU Counter 2		Package
Register Address: C33H, 3123	MSR_PCU_PMON_EVTNSEL3	
Uncore PCU PerfMon Event Select for PCU Counter 3		Package
Register Address: C34H, 3124	MSR_PCU_PMON_BOX_FILTER	
Uncore PCU PerfMon box-wide Filter		Package
Register Address: C36H, 3126	MSR_PCU_PMON_CTR0	
Uncore PCU PerfMon Counter 0		Package
Register Address: C37H, 3127	MSR_PCU_PMON_CTR1	
Uncore PCU PerfMon Counter 1		Package
Register Address: C38H, 3128	MSR_PCU_PMON_CTR2	
Uncore PCU PerfMon Counter 2		Package
Register Address: C39H, 3129	MSR_PCU_PMON_CTR3	
Uncore PCU PerfMon Counter 3		Package
Register Address: D04H, 3332	MSR_CO_PMON_BOX_CTL	
Uncore C-box 0 PerfMon Local Box Wide Control		Package
Register Address: D10H, 3344	MSR_CO_PMON_EVTNSELO	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 0		Package
Register Address: D11H, 3345	MSR_CO_PMON_EVTNSEL1	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 1		Package
Register Address: D12H, 3346	MSR_CO_PMON_EVTNSEL2	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 2		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D13H, 3347	MSR_CO_PMON_EVNTSEL3	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 3		Package
Register Address: D14H, 3348	MSR_CO_PMON_BOX_FILTER	
Uncore C-box 0 PerfMon Box Wide Filter		Package
Register Address: D16H, 3350	MSR_CO_PMON_CTR0	
Uncore C-box 0 PerfMon Counter 0		Package
Register Address: D17H, 3351	MSR_CO_PMON_CTR1	
Uncore C-box 0 PerfMon Counter 1		Package
Register Address: D18H, 3352	MSR_CO_PMON_CTR2	
Uncore C-box 0 PerfMon Counter 2		Package
Register Address: D19H, 3353	MSR_CO_PMON_CTR3	
Uncore C-box 0 PerfMon Counter 3		Package
Register Address: D24H, 3364	MSR_C1_PMON_BOX_CTL	
Uncore C-box 1 PerfMon Local Box Wide Control		Package
Register Address: D30H, 3376	MSR_C1_PMON_EVNTSELO	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 0		Package
Register Address: D31H, 3377	MSR_C1_PMON_EVNTSEL1	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 1		Package
Register Address: D32H, 3378	MSR_C1_PMON_EVNTSEL2	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 2		Package
Register Address: D33H, 3379	MSR_C1_PMON_EVNTSEL3	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 3		Package
Register Address: D34H, 3380	MSR_C1_PMON_BOX_FILTER	
Uncore C-box 1 PerfMon Box Wide Filter		Package
Register Address: D36H, 3382	MSR_C1_PMON_CTR0	
Uncore C-box 1 PerfMon Counter 0		Package
Register Address: D37H, 3383	MSR_C1_PMON_CTR1	
Uncore C-box 1 PerfMon Counter 1		Package
Register Address: D38H, 3384	MSR_C1_PMON_CTR2	
Uncore C-box 1 PerfMon Counter 2		Package
Register Address: D39H, 3385	MSR_C1_PMON_CTR3	
Uncore C-box 1 PerfMon Counter 3		Package
Register Address: D44H, 3396	MSR_C2_PMON_BOX_CTL	
Uncore C-box 2 PerfMon Local Box Wide Control		Package
Register Address: D50H, 3408	MSR_C2_PMON_EVNTSELO	
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 0		Package
Register Address: D51H, 3409	MSR_C2_PMON_EVNTSEL1	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 1		Package
Register Address: D52H, 3410	MSR_C2_PMON_EVNTSEL2	
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 2		Package
Register Address: D53H, 3411	MSR_C2_PMON_EVNTSEL3	
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 3		Package
Register Address: D54H, 3412	MSR_C2_PMON_BOX_FILTER	
Uncore C-box 2 PerfMon Box Wide Filter		Package
Register Address: D56H, 3414	MSR_C2_PMON_CTR0	
Uncore C-box 2 PerfMon Counter 0		Package
Register Address: D57H, 3415	MSR_C2_PMON_CTR1	
Uncore C-box 2 PerfMon Counter 1		Package
Register Address: D58H, 3416	MSR_C2_PMON_CTR2	
Uncore C-box 2 PerfMon Counter 2		Package
Register Address: D59H, 3417	MSR_C2_PMON_CTR3	
Uncore C-box 2 PerfMon Counter 3		Package
Register Address: D64H, 3428	MSR_C3_PMON_BOX_CTL	
Uncore C-box 3 PerfMon Local Box Wide Control		Package
Register Address: D70H, 3440	MSR_C3_PMON_EVNTSEL0	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 0		Package
Register Address: D71H, 3441	MSR_C3_PMON_EVNTSEL1	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 1		Package
Register Address: D72H, 3442	MSR_C3_PMON_EVNTSEL2	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 2		Package
Register Address: D73H, 3443	MSR_C3_PMON_EVNTSEL3	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 3		Package
Register Address: D74H, 3444	MSR_C3_PMON_BOX_FILTER	
Uncore C-box 3 PerfMon Box Wide Filter		Package
Register Address: D76H, 3446	MSR_C3_PMON_CTR0	
Uncore C-box 3 PerfMon Counter 0		Package
Register Address: D77H, 3447	MSR_C3_PMON_CTR1	
Uncore C-box 3 PerfMon Counter 1		Package
Register Address: D78H, 3448	MSR_C3_PMON_CTR2	
Uncore C-box 3 PerfMon Counter 2		Package
Register Address: D79H, 3449	MSR_C3_PMON_CTR3	
Uncore C-box 3 PerfMon Counter 3		Package
Register Address: D84H, 3460	MSR_C4_PMON_BOX_CTL	
Uncore C-box 4 PerfMon Local Box Wide Control		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D90H, 3472	MSR_C4_PMON_EVNTSEL0	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 0		Package
Register Address: D91H, 3473	MSR_C4_PMON_EVNTSEL1	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 1		Package
Register Address: D92H, 3474	MSR_C4_PMON_EVNTSEL2	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 2		Package
Register Address: D93H, 3475	MSR_C4_PMON_EVNTSEL3	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 3		Package
Register Address: D94H, 3476	MSR_C4_PMON_BOX_FILTER	
Uncore C-box 4 PerfMon Box Wide Filter		Package
Register Address: D96H, 3478	MSR_C4_PMON_CTR0	
Uncore C-box 4 PerfMon Counter 0		Package
Register Address: D97H, 3479	MSR_C4_PMON_CTR1	
Uncore C-box 4 PerfMon Counter 1		Package
Register Address: D98H, 3480	MSR_C4_PMON_CTR2	
Uncore C-box 4 PerfMon Counter 2		Package
Register Address: D99H, 3481	MSR_C4_PMON_CTR3	
Uncore C-box 4 PerfMon Counter 3		Package
Register Address: DA4H, 3492	MSR_C5_PMON_BOX_CTL	
Uncore C-box 5 PerfMon Local Box Wide Control		Package
Register Address: DB0H, 3504	MSR_C5_PMON_EVNTSEL0	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 0		Package
Register Address: DB1H, 3505	MSR_C5_PMON_EVNTSEL1	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 1		Package
Register Address: DB2H, 3506	MSR_C5_PMON_EVNTSEL2	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 2		Package
Register Address: DB3H, 3507	MSR_C5_PMON_EVNTSEL3	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 3		Package
Register Address: DB4H, 3508	MSR_C5_PMON_BOX_FILTER	
Uncore C-box 5 PerfMon Box Wide Filter		Package
Register Address: DB6H, 3510	MSR_C5_PMON_CTR0	
Uncore C-box 5 PerfMon Counter 0		Package
Register Address: DB7H, 3511	MSR_C5_PMON_CTR1	
Uncore C-box 5 PerfMon Counter 1		Package
Register Address: DB8H, 3512	MSR_C5_PMON_CTR2	
Uncore C-box 5 PerfMon Counter 2		Package
Register Address: DB9H, 3513	MSR_C5_PMON_CTR3	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 5 PerfMon Counter 3		Package
Register Address: DC4H, 3524	MSR_C6_PMON_BOX_CTL	
Uncore C-box 6 PerfMon Local Box Wide Control		Package
Register Address: DD0H, 3536	MSR_C6_PMON_EVTNTSEL0	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 0		Package
Register Address: DD1H, 3537	MSR_C6_PMON_EVTNTSEL1	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 1		Package
Register Address: DD2H, 3538	MSR_C6_PMON_EVTNTSEL2	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 2		Package
Register Address: DD3H, 3539	MSR_C6_PMON_EVTNTSEL3	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 3		Package
Register Address: DD4H, 3540	MSR_C6_PMON_BOX_FILTER	
Uncore C-box 6 PerfMon Box Wide Filter		Package
Register Address: DD6H, 3542	MSR_C6_PMON_CTR0	
Uncore C-box 6 PerfMon Counter 0		Package
Register Address: DD7H, 3543	MSR_C6_PMON_CTR1	
Uncore C-box 6 PerfMon Counter 1		Package
Register Address: DD8H, 3544	MSR_C6_PMON_CTR2	
Uncore C-box 6 PerfMon Counter 2		Package
Register Address: DD9H, 3545	MSR_C6_PMON_CTR3	
Uncore C-box 6 PerfMon Counter 3		Package
Register Address: DE4H, 3556	MSR_C7_PMON_BOX_CTL	
Uncore C-box 7 PerfMon Local Box Wide Control		Package
Register Address: DF0H, 3568	MSR_C7_PMON_EVTNTSEL0	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 0		Package
Register Address: DF1H, 3569	MSR_C7_PMON_EVTNTSEL1	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 1		Package
Register Address: DF2H, 3570	MSR_C7_PMON_EVTNTSEL2	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 2		Package
Register Address: DF3H, 3571	MSR_C7_PMON_EVTNTSEL3	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 3		Package
Register Address: DF4H, 3572	MSR_C7_PMON_BOX_FILTER	
Uncore C-box 7 PerfMon Box Wide Filter		Package
Register Address: DF6H, 3574	MSR_C7_PMON_CTR0	
Uncore C-box 7 PerfMon Counter 0		Package
Register Address: DF7H, 3575	MSR_C7_PMON_CTR1	
Uncore C-box 7 PerfMon Counter 1		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DF8H, 3576	MSR_C7_PMON_CTR2	
Uncore C-box 7 PerfMon Counter 2		Package
Register Address: DF9H, 3577	MSR_C7_PMON_CTR3	
Uncore C-box 7 PerfMon Counter 3		Package

2.12 MSRS IN THE 3RD GENERATION INTEL® CORE™ PROCESSOR FAMILY BASED ON IVY BRIDGE MICROARCHITECTURE

The 3rd generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200v2 product family based on Ivy Bridge microarchitecture support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, and Table 2-25. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates that TDP Limit for Turbo mode is not programmable.	Package
31:30	Reserved.	
32	Low Power Mode Support (LPM) (R/O) When set to 1, indicates that LPM is supported. When set to 0, indicates LPM is not supported.	Package
34:33	Number of ConfigTDP Levels (R/O) 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 03: Reserved	Package
39:35	Reserved.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
55:48	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.	Package
63:56	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .	Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
24:16	Reserved	
25	C3 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	
26	C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Enable C3 Undemotion (R/W) When set, enables undemotion from demoted C3.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
28	Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.	
63:29	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O)		Package
7:0	Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).	
63:8	Reserved.	
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
	ConfigTDP Level 1 ratio and power level (R/O)	Package
14:0	PKG_TDP_LVL1 Power setting for ConfigTDP Level 1.	
15	Reserved.	
23:16	Config_TDP_LVL1_Ratio ConfigTDP level 1 ratio to be used for this specific processor.	
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL1 Max Power setting allowed for ConfigTDP Level 1.	
47	Reserved.	
62:48	PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.	
63	Reserved.	
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 ratio and power level (R/O)		Package
14:0	PKG_TDP_LVL2 Power setting for ConfigTDP Level 2.	
15	Reserved.	
23:16	Config_TDP_LVL2_Ratio ConfigTDP level 2 ratio to be used for this specific processor.	
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL2 Max Power setting allowed for ConfigTDP Level 2.	
47	Reserved.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
62:48	PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.	
63	Reserved.	
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W)		Package
1:0	TDP_LEVEL (RW/L) System BIOS can program this field.	
30:2	Reserved.	
31	Config_TDP_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W)		Package
7:0	MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.	
30:8	Reserved.	
31	TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
See Table 2-20, Table 2-21, and Table 2-22 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.		

2.12.1 MSRs in the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture

Table 2-26 lists model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 v2 Product Family (based on Ivy Bridge-E microarchitecture). These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH; see Table 2-1. These processors supports the MSR interfaces listed in Table 2-20 and Table 2-26.

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/W0) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved.	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
22:16	Reserved.	
23	PPIN_CAP (R/O) When set to 1, indicates that Protected Processor Inventory Number (PPIN) capability can be enabled for a privileged system inventory agent to read PPIN from MSR_PPIN. When set to 0, PPIN capability is not supported. An attempt to access MSR_PPIN_CTL or MSR_PPIN will cause #GP.	Package
27:24	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
30	Programmable TJ OFFSET (R/O) When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify a temperature offset.	Package
39:31	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	Reserved.	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17FH, 383	MSR_ERROR_CONTROL	
MC Bank Error Configuration (R/W)		Package
0	Reserved.	
1	MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.	
63:2	Reserved.	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R/O) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
27:24	TCC Activation Offset (R/W) Specifies a temperature offset in degrees C from the temperature target (bits 23:16). PROCHOT# will assert at the offset target temperature. Write is permitted only if MSR_PLATFORM_INFO.[30] is set.	
63:28	Reserved.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package
63:32	Reserved.	
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 296H, 662	IA32_MC22_CTL2	
See Table 2-2.		Package
Register Address: 297H, 663	IA32_MC23_CTL2IA32_MC23_CTL2	
See Table 2-2.		Package
Register Address: 298H, 664	IA32_MC24_CTL2	
See Table 2-2.		Package
Register Address: 299H, 665	IA32_MC25_CTL2	
See Table 2-2.		Package
Register Address: 29AH, 666	IA32_MC26_CTL2	
See Table 2-2.		Package
Register Address: 29BH, 667	IA32_MC27_CTL2	
See Table 2-2.		Package
Register Address: 29CH, 668	IA32_MC28_CTL2	
See Table 2-2.		Package
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 454H, 1108	IA32_MC21_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 458H, 1112	IA32_MC22_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 459H, 1113	IA32_MC22_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45AH, 1114	IA32_MC22_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45BH, 1115	IA32_MC22_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45CH, 1116	IA32_MC23_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45DH, 1117	IA32_MC23_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45EH, 1118	IA32_MC23_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 45FH, 1119	IA32_MC23_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 460H, 1120	IA32_MC24_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 461H, 1121	IA32_MC24_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 462H, 1122	IA32_MC24_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 463H, 1123	IA32_MC24_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 464H, 1124	IA32_MC25_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 465H, 1125	IA32_MC25_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 466H, 1126	IA32_MC25_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 467H, 1127	IA32_MC2MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 468H, 1128	IA32_MC26_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 469H, 1129	IA32_MC26_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46AH, 1130	IA32_MC26_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46BH, 1131	IA32_MC26_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46CH, 1132	IA32_MC27_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46DH, 1133	IA32_MC27_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46EH, 1134	IA32_MC27_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 46FH, 1135	IA32_MC27_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 470H, 1136	IA32_MC28_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 471H, 1137	IA32_MC28_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 472H, 1138	IA32_MC28_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 473H, 1139	IA32_MC28_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
Package RAPL Perf Status (R/O)		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
See Table 2-20, for other MSR definitions applicable to Intel Xeon processor E5 v2 with a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH.		

2.12.2 Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family

The Intel® Xeon® processor E7 v2 family (based on Ivy Bridge-E microarchitecture) with a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH supports the MSR interfaces listed in Table 2-20, Table 2-26, and Table 2-27.

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	
2	Enable VMX Outside SMX Operation (R/WL)	
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
63:16	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
63:25	Reserved.	
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status (R/W)		Thread
0	RIPV	
1	EIPV	
2	MCIP	
3	LMCE Signaled	
63:4	Reserved.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
39:32	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.	Package
47:40	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.	Package
55:48	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.	Package
62:56	Reserved.	
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT and MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 29DH, 669	IA32_MC29_CTL2	
See Table 2-2.		Package
Register Address: 29EH, 670	IA32_MC30_CTL2	
See Table 2-2.		Package
Register Address: 29FH, 671	IA32_MC31_CTL2	
See Table 2-2.		Package
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 22.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
$n:0$	Enable PEBS on IA32_PMCx. (R/W)	
$31:n+1$	Reserved.	
$32+m:32$	Enable Load Latency on IA32_PMCx. (R/W)	
$63:33+m$	Reserved.	
Register Address: 41BH, 1051	IA32_MC6_MISC	
Misc MAC Information of Integrated I/O (R/O) See Section 18.3.2.4.		Package
5:0	Recoverable Address LSB	
8:6	Address Mode	
15:9	Reserved.	
31:16	PCI Express Requestor ID	
39:32	PCI Express Segment Number	
63:32	Reserved.	
Register Address: 474H, 1140	IA32_MC29_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 475H, 1141	IA32_MC29_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 476H, 1142	IA32_MC29_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 477H, 1143	IA32_MC29_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 478H, 1144	IA32_MC30_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 479H, 1145	IA32_MC30_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47AH, 1146	IA32_MC30_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47BH, 1147	IA32_MC30_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47CH, 1148	IA32_MC31_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47DH, 1149	IA32_MC31_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47EH, 1150	IA32_MC31_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47FH, 1147	IA32_MC31_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
See Table 2-20, Table 2-26 for other MSR definitions applicable to Intel Xeon processor E7 v2 with a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.12.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families

Intel Xeon Processor E5 v2 and E7 v2 families are based on the Ivy Bridge-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-24 and Table 2-28. For complete detail of the uncore PMU, refer to Intel

Xeon Processor E5 v2 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH.

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C00H, 3072	MSR_PMON_GLOBAL_CTL	
Uncore PerfMon Per-Socket Global Control		Package
Register Address: C01H, 3073	MSR_PMON_GLOBAL_STATUS	
Uncore PerfMon Per-Socket Global Status		Package
Register Address: C06H, 3078	MSR_PMON_GLOBAL_CONFIG	
Uncore PerfMon Per-Socket Global Configuration		Package
Register Address: C15H, 3093	MSR_U_PMON_BOX_STATUS	
Uncore U-box PerfMon U-Box Wide Status		Package
Register Address: C35H, 3125	MSR_PCU_PMON_BOX_STATUS	
Uncore PCU PerfMon Box Wide Status		Package
Register Address: D1AH, 3354	MSR_C0_PMON_BOX_FILTER1	
Uncore C-Box 0 PerfMon Box Wide Filter1		Package
Register Address: D3AH, 3386	MSR_C1_PMON_BOX_FILTER1	
Uncore C-Box 1 PerfMon Box Wide Filter1		Package
Register Address: D5AH, 3418	MSR_C2_PMON_BOX_FILTER1	
Uncore C-Box 2 PerfMon Box Wide Filter1		Package
Register Address: D7AH, 3450	MSR_C3_PMON_BOX_FILTER1	
Uncore C-Box 3 PerfMon Box Wide Filter1		Package
Register Address: D9AH, 3482	MSR_C4_PMON_BOX_FILTER1	
Uncore C-Box 4 PerfMon Box Wide Filter1		Package
Register Address: DBAH, 3514	MSR_C5_PMON_BOX_FILTER1	
Uncore C-Box 5 PerfMon Box Wide Filter1		Package
Register Address: DDAH, 3546	MSR_C6_PMON_BOX_FILTER1	
Uncore C-Box 6 PerfMon Box Wide Filter1		Package
Register Address: DFAH, 3578	MSR_C7_PMON_BOX_FILTER1	
Uncore C-Box 7 PerfMon Box Wide Filter1		Package
Register Address: E04H, 3588	MSR_C8_PMON_BOX_CTL	
Uncore C-Box 8 PerfMon Local Box Wide Control		Package
Register Address: E10H, 3600	MSR_C8_PMON_EVNTSELO	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 0		Package
Register Address: E11H, 3601	MSR_C8_PMON_EVNTSEL1	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 1		Package
Register Address: E12H, 3602	MSR_C8_PMON_EVNTSEL2	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 2		Package
Register Address: E13H, 3603	MSR_C8_PMON_EVNTSEL3	

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 3		Package
Register Address: E14H, 3604	MSR_C8_PMON_BOX_FILTER	
Uncore C-Box 8 PerfMon Box Wide Filter		Package
Register Address: E16H, 3606	MSR_C8_PMON_CTR0	
Uncore C-Box 8 PerfMon Counter 0		Package
Register Address: E17H, 3607	MSR_C8_PMON_CTR1	
Uncore C-Box 8 PerfMon Counter 1		Package
Register Address: E18H, 3608	MSR_C8_PMON_CTR2	
Uncore C-Box 8 PerfMon Counter 2		Package
Register Address: E19H, 3609	MSR_C8_PMON_CTR3	
Uncore C-Box 8 PerfMon Counter 3		Package
Register Address: E1AH, 3610	MSR_C8_PMON_BOX_FILTER1	
Uncore C-Box 8 PerfMon Box Wide Filter1		Package
Register Address: E24H, 3620	MSR_C9_PMON_BOX_CTL	
Uncore C-Box 9 PerfMon Local Box Wide Control		Package
Register Address: E30H, 3632	MSR_C9_PMON_EVNTSEL0	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 0		Package
Register Address: E31H, 3633	MSR_C9_PMON_EVNTSEL1	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 1		Package
Register Address: E32H, 3634	MSR_C9_PMON_EVNTSEL2	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 2		Package
Register Address: E33H, 3635	MSR_C9_PMON_EVNTSEL3	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 3		Package
Register Address: E34H, 3636	MSR_C9_PMON_BOX_FILTER	
Uncore C-Box 9 PerfMon Box Wide Filter		Package
Register Address: E36H, 3638	MSR_C9_PMON_CTR0	
Uncore C-Box 9 PerfMon Counter 0		Package
Register Address: E37H, 3639	MSR_C9_PMON_CTR1	
Uncore C-Box 9 PerfMon Counter 1		Package
Register Address: E38H, 3640	MSR_C9_PMON_CTR2	
Uncore C-Box 9 PerfMon Counter 2		Package
Register Address: E39H, 3641	MSR_C9_PMON_CTR3	
Uncore C-Box 9 PerfMon Counter 3		Package
Register Address: E3AH, 3642	MSR_C9_PMON_BOX_FILTER1	
Uncore C-Box 9 PerfMon Box Wide Filter1		Package
Register Address: E44H, 3652	MSR_C10_PMON_BOX_CTL	
Uncore C-Box 10 PerfMon Local Box Wide Control		Package

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E50H, 3664	MSR_C10_PMON_EVTNSELO	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 0		Package
Register Address: E51H, 3665	MSR_C10_PMON_EVTNSEL1	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 1		Package
Register Address: E52H, 3666	MSR_C10_PMON_EVTNSEL2	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 2		Package
Register Address: E53H, 3667	MSR_C10_PMON_EVTNSEL3	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 3		Package
Register Address: E54H, 3668	MSR_C10_PMON_BOX_FILTER	
Uncore C-Box 10 PerfMon Box Wide Filter		Package
Register Address: E56H, 3670	MSR_C10_PMON_CTR0	
Uncore C-Box 10 PerfMon Counter 0		Package
Register Address: E57H, 3671	MSR_C10_PMON_CTR1	
Uncore C-Box 10 PerfMon Counter 1		Package
Register Address: E58H, 3672	MSR_C10_PMON_CTR2	
Uncore C-Box 10 PerfMon Counter 2		Package
Register Address: E59H, 3673	MSR_C10_PMON_CTR3	
Uncore C-Box 10 PerfMon Counter 3		Package
Register Address: E5AH, 3674	MSR_C10_PMON_BOX_FILTER1	
Uncore C-Box 10 PerfMon Box Wide Filter1		Package
Register Address: E64H, 3684	MSR_C11_PMON_BOX_CTL	
Uncore C-Box 11 PerfMon Local Box Wide Control		Package
Register Address: E70H, 3696	MSR_C11_PMON_EVTNSELO	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 0		Package
Register Address: E71H, 3697	MSR_C11_PMON_EVTNSEL1	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 1		Package
Register Address: E72H, 3698	MSR_C11_PMON_EVTNSEL2	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 2		Package
Register Address: E73H, 3699	MSR_C11_PMON_EVTNSEL3	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 3		Package
Register Address: E74H, 3700	MSR_C11_PMON_BOX_FILTER	
Uncore C-Box 11 PerfMon Box Wide Filter		Package
Register Address: E76H, 3702	MSR_C11_PMON_CTR0	
Uncore C-Box 11 PerfMon Counter 0		Package
Register Address: E77H, 3703	MSR_C11_PMON_CTR1	
Uncore C-Box 11 PerfMon Counter 1		Package
Register Address: E78H, 3704	MSR_C11_PMON_CTR2	

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 11 PerfMon Counter 2		Package
Register Address: E79H, 3705	MSR_C11_PMON_CTR3	
Uncore C-Box 11 PerfMon Counter 3		Package
Register Address: E7AH, 3706	MSR_C11_PMON_BOX_FILTER1	
Uncore C-Box 11 PerfMon Box Wide Filter1		Package
Register Address: E84H, 3716	MSR_C12_PMON_BOX_CTL	
Uncore C-Box 12 PerfMon Local Box Wide Control		Package
Register Address: E90H, 3728	MSR_C12_PMON_EVNTSEL0	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 0		Package
Register Address: E91H, 3729	MSR_C12_PMON_EVNTSEL1	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 1		Package
Register Address: E92H, 3730	MSR_C12_PMON_EVNTSEL2	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 2		Package
Register Address: E93H, 3731	MSR_C12_PMON_EVNTSEL3	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 3		Package
Register Address: E94H, 3732	MSR_C12_PMON_BOX_FILTER	
Uncore C-Box 12 PerfMon Box Wide Filter		Package
Register Address: E96H, 3734	MSR_C12_PMON_CTR0	
Uncore C-Box 12 PerfMon Counter 0		Package
Register Address: E97H, 3735	MSR_C12_PMON_CTR1	
Uncore C-Box 12 PerfMon Counter 1		Package
Register Address: E98H, 3736	MSR_C12_PMON_CTR2	
Uncore C-Box 12 PerfMon Counter 2		Package
Register Address: E99H, 3737	MSR_C12_PMON_CTR3	
Uncore C-Box 12 PerfMon Counter 3		Package
Register Address: E9AH, 3738	MSR_C12_PMON_BOX_FILTER1	
Uncore C-Box 12 PerfMon Box Wide Filter1		Package
Register Address: EA4H, 3748	MSR_C13_PMON_BOX_CTL	
Uncore C-Box 13 PerfMon Local Box Wide Control		Package
Register Address: EB0H, 3760	MSR_C13_PMON_EVNTSEL0	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 0		Package
Register Address: EB1H, 3761	MSR_C13_PMON_EVNTSEL1	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 1		Package
Register Address: EB2H, 3762	MSR_C13_PMON_EVNTSEL2	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 2		Package
Register Address: EB3H, 3763	MSR_C13_PMON_EVNTSEL3	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 3		Package

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: EB4H, 3764	MSR_C13_PMON_BOX_FILTER	
Uncore C-Box 13 PerfMon Box Wide Filter		Package
Register Address: EB6H, 3766	MSR_C13_PMON_CTR0	
Uncore C-Box 13 PerfMon Counter 0		Package
Register Address: EB7H, 3767	MSR_C13_PMON_CTR1	
Uncore C-Box 13 PerfMon Counter 1		Package
Register Address: EB8H, 3768	MSR_C13_PMON_CTR2	
Uncore C-Box 13 PerfMon Counter 2		Package
Register Address: EB9H, 3769	MSR_C13_PMON_CTR3	
Uncore C-Box 13 PerfMon Counter 3		Package
Register Address: EBAH, 3770	MSR_C13_PMON_BOX_FILTER1	
Uncore C-Box 13 PerfMon Box Wide Filter1		Package
Register Address: EC4H, 3780	MSR_C14_PMON_BOX_CTL	
Uncore C-Box 14 PerfMon Local Box Wide Control		Package
Register Address: ED0H, 3792	MSR_C14_PMON_EVTSELO	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 0		Package
Register Address: ED1H, 3793	MSR_C14_PMON_EVTSEL1	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 1		Package
Register Address: ED2H, 3794	MSR_C14_PMON_EVTSEL2	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 2		Package
Register Address: ED3H, 3795	MSR_C14_PMON_EVTSEL3	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 3		Package
Register Address: ED4H, 3796	MSR_C14_PMON_BOX_FILTER	
Uncore C-Box 14 PerfMon Box Wide Filter		Package
Register Address: ED6H, 3798	MSR_C14_PMON_CTR0	
Uncore C-Box 14 PerfMon Counter 0		Package
Register Address: ED7H, 3799	MSR_C14_PMON_CTR1	
Uncore C-Box 14 PerfMon Counter 1		Package
Register Address: ED8H, 3800	MSR_C14_PMON_CTR2	
Uncore C-Box 14 PerfMon Counter 2		Package
Register Address: ED9H, 3801	MSR_C14_PMON_CTR3	
Uncore C-Box 14 PerfMon Counter 3		Package
Register Address: EDAH, 3802	MSR_C14_PMON_BOX_FILTER1	
Uncore C-Box 14 PerfMon Box Wide Filter1		Package

2.13 MSRS IN THE 4TH GENERATION INTEL® CORE™ PROCESSORS BASED ON HASWELL MICROARCHITECTURE

The 4th generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200v3 product family (based on Haswell microarchitecture), with a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH, 06_45H, or 06_46H, support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, and Table 2-29. For an MSR listed in Table 2-20 that also appears in Table 2-29, Table 2-29 supersedes Table 2-20.

The MSRs listed in Table 2-29 also apply to processors based on Haswell-E microarchitecture (see Section 2.14).

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
31:30	Reserved.	
32	Low Power Mode Support (LPM) (R/O) When set to 1, indicates that LPM is supported. When set to 0, indicates LPM is not supported.	Package
34:33	Number of ConfigTDP Levels (R/O) 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 03: Reserved.	Package
39:35	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
55:48	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.	Package
63:56	Reserved.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 186H, 390	IA32_PERFEVTSEL0	
Performance Event Select for Counter 0 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 22.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
Performance Event Select for Counter 1 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 22.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
Register Address: 188H, 392	IA32_PERFEVTSEL2	
Performance Event Select for Counter 2 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 22.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
33	IN_TXCP: See Section 22.3.6.5.1. When IN_TXCP=1 & IN_TX=1 and in sampling, a spurious PMI may occur and transactions may continuously abort near overflow conditions. Software should favor using IN_TXCP for counting over sampling. If sampling, software should use large "sample-after" value after clearing the counter configured to use IN_TXCP and also always reset the counter even when no overflow condition was reported.	
Register Address: 189H, 393	IA32_PERFEVTSEL3	
Performance Event Select for Counter 3 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 22.3.6.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W)		Thread
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
8	FAR_BRANCH	
9	EN_CALL_STACK	
63:9	Reserved.	
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
0	LBR: Last Branch Record	
1	BTF	
5:2	Reserved.	
6	TR: Branch Trace	
7	BTS: Log Branch Trace Message to BTS Buffer	
8	BTINT	
9	BTS_OFF_OS	
10	BTS_OFF_USER	
11	FREEZE_LBR_ON_PMI	
12	FREEZE_PERFMON_ON_PMI	
13	ENABLE_UNCORE_PMI	
14	FREEZE_WHILE_SMM	
15	RTM_DEBUG	
63:15	Reserved.	
Register Address: 491H, 1169	IA32_VMX_VMFUNC	
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Thread
Register Address: 60BH, 1548	MSR_PKG_C_IRT_L1	
Package C6/C7 Interrupt Response Limit 1 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7 state. The latency programmed in this register is for the shorter-latency sub C-states used by an MWAIT hint to a C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:16	Reserved.	
Register Address: 60CH, 1548	MSR_PKG_C6_C7_INTERRUPT_RESPONSE_LIMIT_2	
Package C6/C7 Interrupt Response Limit 2 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7 state. The latency programmed in this register is for the longer-latency sub C-states used by an MWAIT hint to a C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt response time limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O)		Package
7:0	Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).	
63:8	Reserved.	
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
ConfigTDP Level 1 Ratio and Power Level (R/O)		Package
14:0	PKG_TDP_LVL1 Power setting for ConfigTDP Level 1.	
15	Reserved.	
23:16	Config_TDP_LVL1_Ratio ConfigTDP level 1 ratio to be used for this specific processor.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL1 Max Power setting allowed for ConfigTDP Level 1.	
62:47	PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.	
63	Reserved.	
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 Ratio and Power Level (R/O)		Package
14:0	PKG_TDP_LVL2 Power setting for ConfigTDP Level 2.	
15	Reserved.	
23:16	Config_TDP_LVL2_Ratio ConfigTDP level 2 ratio to be used for this specific processor.	
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL2 Max Power setting allowed for ConfigTDP Level 2.	
62:47	PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.	
63	Reserved.	
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W)		Package
1:0	TDP_LEVEL (RW/L) System BIOS can program this field.	
30:2	Reserved.	
31	Config_TDP_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W)		Package
7:0	MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.	
30:8	Reserved.	
31	TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: C80H, 3200	IA32_DEBUG_INTERFACE	
Silicon Debug Feature Control (R/W) See Table 2-2.		Package

2.13.1 MSRs in the 4th Generation Intel® Core™ Processor Family Based on Haswell Microarchitecture

Table 2-30 lists model-specific registers (MSRs) that are specific to the 4th generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200 v3 product family (based on Haswell microarchitecture). These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH, 06_45H, or 06_46H; see Table 2-1.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s Package C states C7 are not available to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH.	
9:4	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved	
15	CFG Lock (R/WO)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
63:29	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.	
63:60	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 391H, 913	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Core 0 select.	
1	Core 1 select.	
2	Core 2 select.	
3	Core 3 select.	
18:4	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: 392H, 914	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Encoded number of C-Box, derive value by "-1".	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSELO	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb Unit, Counter 1 Event Select MSR		Package
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.		Package
0	Lock (SMM-RW0) When set to '1' locks this register from further changes.	
1	Reserved.	
2	SMM_Code_Chk_En (SMM-RW) This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.	
63:3	Reserved.	
Register Address: 4E2H, 1250	MSR_SMM_DELAYED	
SMM Delayed (SMM-RO) Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.		Package

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a logical processor of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 4E3H, 1251	MSR_SMM_BLOCKED	
SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.		Package
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a logical processor of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is 0FFFH. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 17.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 17.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 640H, 1600	MSR_PP1_POWER_LIMIT	
PP1 RAPL Power Limit Control (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
PP1 Energy Status (R/O) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 642H, 1602	MSR_PP1_POLICY	
PP1 Balance Policy (R/W) See Section 17.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	Graphics Driver Status (R0) When set, frequency is reduced below the operating system request due to Processor Graphics driver override.	
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Core Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.	
10	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
11	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
12	Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
15:14	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
27	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 6B0H, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	Graphics Driver Status (R0) When set, frequency is reduced below the operating system request due to Processor Graphics driver override.	
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Graphics Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
10	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
11	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
15:12	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
27	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
5:2	Reserved.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Reserved.	
10	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
11	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
15:12	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:30	Reserved.	
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1824	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
See Table 2-20, Table 2-21, Table 2-22, Table 2-25, and Table 2-29 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 063CH or 06_46H.		

2.13.2 Additional Residency MSRs Supported in 4th Generation Intel® Core™ Processors

The 4th generation Intel® Core™ processor family (based on Haswell microarchitecture) with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H supports the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-29, Table 2-30, and Table 2-31.

Table 2-31. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_45H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10	
9:4	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/WO)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
63:29	Reserved.	
Register Address: 630H, 1584	MSR_PKG_C8_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C8 Residency Counter (R/O) Value since last reset that this package is in processor-specific C8 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 631H, 1585	MSR_PKG_C9_RESIDENCY	

Table 2-31. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_45H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C9 Residency Counter (R/O) Value since last reset that this package is in processor-specific C9 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 632H, 1586	MSR_PKG_C10_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C10 Residency Counter (R/O) Value since last reset that this package is in processor-specific C10 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
See Table 2-20, Table 2-21, Table 2-22, Table 2-29, and Table 2-30 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H.		

2.14 MSRS IN THE INTEL® XEON® PROCESSOR E5 V3 AND E7 V3 PRODUCT FAMILY

The Intel® Xeon® processor E5 v3 family and the Intel® Xeon® processor E7 v3 family are based on Haswell-E microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_3F). These processors support the MSR interfaces listed in Table 2-20, Table 2-29, and Table 2-32.

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 35H, 53	MSR_CORE_THREAD_COUNT	
Configured State of Enabled Processor Core Count and Logical Processor Count (R/O)		Package
- After a Power-On RESET, enumerates factory configuration of the number of processor cores and logical processors in the physical package. - Following the sequence of (i) BIOS modified a Configuration Mask which selects a subset of processor cores to be active post RESET and (ii) a RESET event after the modification, enumerates the current configuration of enabled processor core count and logical processor count in the physical package.		
15:0	THREAD_COUNT (R/O) The number of logical processors that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.	
31:16	Core_COUNT (R/O) The number of processor cores that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.	
63:32	Reserved.	
Register Address: 53H, 83	MSR_THREAD_ID_INFO	
A Hardware Assigned ID for the Logical Processor (R/O)		Thread

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
7:0	Logical_Processor_ID (R/O) An implementation-specific numerical value physically assigned to each logical processor. This ID is not related to Initial APIC ID or x2APIC ID, it is unique within a physical package.	
63:8	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/WO)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	MCG_EM_P	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.	
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.	
63:60	Reserved.	
Register Address: 17FH, 383	MSR_ERROR_CONTROL	
MC Bank Error Configuration (R/W)		Package
0	Reserved.	
1	MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.	
63:2	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.	Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.	Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package
39:32	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.	Package
47:40	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.	Package
55:48	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.	Package
63:56	Maximum Ratio Limit for 16C Maximum turbo ratio limit of 16 core active.	Package
Register Address: 1AFH, 431	MSR_TURBO_RATIO_LIMIT2	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 17C Maximum turbo ratio limit of 17 core active.	Package
15:8	Maximum Ratio Limit for 18C Maximum turbo ratio limit of 18 core active.	Package
62:16	Reserved.	Package
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1, and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.	Package	
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 454H, 1108	IA32_MC21_CTL	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 17.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 17.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy Consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61EH, 1566	MSR_PCIE_PLL_RATIO	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Configuration of PCIE PLL Relative to BCLK(R/W)		Package
1:0	PCIE Ratio (R/W) 00b: Use 5:5 mapping for 100MHz operation (default). 01b: Use 5:4 mapping for 125MHz operation. 10b: Use 5:3 mapping for 166MHz operation. 11b: Use 5:2 mapping for 250MHz operation.	Package
2	LPLL Select (R/W) If 1, use configured setting of PCIE Ratio.	Package
3	LONG RESET (R/W) If 1, wait an additional time-out before re-locking Gen2/Gen3 PLLs.	Package
63:4	Reserved.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
2	Power Budget Management Status (R0) When set, frequency is reduced below the operating system request due to PBM limit	
3	Platform Configuration Services Status (R0) When set, frequency is reduced below the operating system request due to PCS limit	
4	Reserved.	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Reserved.	
10	Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits.	
12:11	Reserved.	
13	Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1.	
14	Core Max N-Core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency.	
15	Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
18	Power Budget Management Log When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19	Platform Configuration Services Log When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20	Reserved.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Multi-Core Turbo Log When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28:27	Reserved.	
29	Core Frequency P1 Log When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
30	Core Max N-Core Turbo Frequency Limiting Log When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
31	Core Frequency Limiting Log When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:32	Reserved.	
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread
7:0	EventID (R/W) Event encoding: 0x0: No monitoring. 0x1: L3 occupancy monitoring. All other encoding reserved.	
31:8	Reserved.	
41:32	RMID (R/W)	
63:42	Reserved.	
Register Address: C8EH, 3214	IA32_QM_CTR	
Monitoring Counter Register (R/O) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
61:0	Resource Monitored Data	
62	Unavailable: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.	
63	Error: If 1, indicates an unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
9:0	RMID	
63: 10	Reserved.	
See Table 2-20 and Table 2-29 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3FH.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.14.1 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family

The Intel Xeon Processor E5 v3 and E7 v3 families are based on Haswell-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-33. For complete details of the uncore PMU, refer to the Intel Xeon Processor E5 v3 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3FH.

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 700H, 1792	MSR_PMON_GLOBAL_CTL	
Uncore PerfMon Per-Socket Global Control		Package
Register Address: 701H, 1793	MSR_PMON_GLOBAL_STATUS	
Uncore PerfMon Per-Socket Global Status		Package
Register Address: 702H, 1794	MSR_PMON_GLOBAL_CONFIG	
Uncore PerfMon Per-Socket Global Configuration		Package
Register Address: 703H, 1795	MSR_U_PMON_UCLK_FIXED_CTL	
Uncore U-Box UCLK Fixed Counter Control		Package
Register Address: 704H, 1796	MSR_U_PMON_UCLK_FIXED_CTR	
Uncore U-Box UCLK Fixed Counter		Package
Register Address: 705H, 1797	MSR_U_PMON_EVNTSELO	
Uncore U-Box PerfMon Event Select for U-Box Counter 0		Package
Register Address: 706H, 1798	MSR_U_PMON_EVNTSEL1	
Uncore U-Box PerfMon Event Select for U-Box Counter 1		Package
Register Address: 708H, 1800	MSR_U_PMON_BOX_STATUS	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore U-Box PerfMon U-Box Wide Status		Package
Register Address: 709H, 1801	MSR_U_PMON_CTRL0	
Uncore U-Box PerfMon Counter 0		Package
Register Address: 70AH, 1802	MSR_U_PMON_CTRL1	
Uncore U-Box PerfMon Counter 1		Package
Register Address: 710H, 1808	MSR_PCU_PMON_BOX_CTL	
Uncore PCU PerfMon for PCU-Box-Wide Control		Package
Register Address: 711H, 1809	MSR_PCU_PMON_EVTSEL0	
Uncore PCU PerfMon Event Select for PCU Counter 0		Package
Register Address: 712H, 1810	MSR_PCU_PMON_EVTSEL1	
Uncore PCU PerfMon Event Select for PCU Counter 1		Package
Register Address: 713H, 1811	MSR_PCU_PMON_EVTSEL2	
Uncore PCU PerfMon Event Select for PCU Counter 2		Package
Register Address: 714H, 1812	MSR_PCU_PMON_EVTSEL3	
Uncore PCU PerfMon Event Select for PCU Counter 3		Package
Register Address: 715H, 1813	MSR_PCU_PMON_BOX_FILTER	
Uncore PCU PerfMon Box-Wide Filter		Package
Register Address: 716H, 1814	MSR_PCU_PMON_BOX_STATUS	
Uncore PCU PerfMon Box Wide Status		Package
Register Address: 717H, 1815	MSR_PCU_PMON_CTRL0	
Uncore PCU PerfMon Counter 0		Package
Register Address: 718H, 1816	MSR_PCU_PMON_CTRL1	
Uncore PCU PerfMon Counter 1		Package
Register Address: 719H, 1817	MSR_PCU_PMON_CTRL2	
Uncore PCU PerfMon Counter 2		Package
Register Address: 71AH, 1818	MSR_PCU_PMON_CTRL3	
Uncore PCU PerfMon Counter 3		Package
Register Address: 720H, 1824	MSR_S0_PMON_BOX_CTL	
Uncore SBo 0 PerfMon for SBo 0 Box-Wide Control		Package
Register Address: 721H, 1825	MSR_S0_PMON_EVTSEL0	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 0		Package
Register Address: 722H, 1826	MSR_S0_PMON_EVTSEL1	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 1		Package
Register Address: 723H, 1827	MSR_S0_PMON_EVTSEL2	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 2		Package
Register Address: 724H, 1828	MSR_S0_PMON_EVTSEL3	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 3		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 725H, 1829	MSR_S0_PMON_BOX_FILTER	
Uncore SBo 0 PerfMon Box-Wide Filter		Package
Register Address: 726H, 1830	MSR_S0_PMON_CTR0	
Uncore SBo 0 PerfMon Counter 0		Package
Register Address: 727H, 1831	MSR_S0_PMON_CTR1	
Uncore SBo 0 PerfMon Counter 1		Package
Register Address: 728H, 1832	MSR_S0_PMON_CTR2	
Uncore SBo 0 PerfMon Counter 2		Package
Register Address: 729H, 1833	MSR_S0_PMON_CTR3	
Uncore SBo 0 PerfMon Counter 3		Package
Register Address: 72AH, 1834	MSR_S1_PMON_BOX_CTL	
Uncore SBo 1 PerfMon for SBo 1 Box-Wide Control		Package
Register Address: 72BH, 1835	MSR_S1_PMON_EVTNSELO	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 0		Package
Register Address: 72CH, 1836	MSR_S1_PMON_EVTNSEL1	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 1		Package
Register Address: 72DH, 1837	MSR_S1_PMON_EVTNSEL2	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 2		Package
Register Address: 72EH, 1838	MSR_S1_PMON_EVTNSEL3	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 3		Package
Register Address: 72FH, 1839	MSR_S1_PMON_BOX_FILTER	
Uncore SBo 1 PerfMon Box-Wide Filter		Package
Register Address: 730H, 1840	MSR_S1_PMON_CTR0	
Uncore SBo 1 PerfMon Counter 0		Package
Register Address: 731H, 1841	MSR_S1_PMON_CTR1	
Uncore SBo 1 PerfMon Counter 1		Package
Register Address: 732H, 1842	MSR_S1_PMON_CTR2	
Uncore SBo 1 PerfMon Counter 2		Package
Register Address: 733H, 1843	MSR_S1_PMON_CTR3	
Uncore SBo 1 PerfMon Counter 3		Package
Register Address: 734H, 1844	MSR_S2_PMON_BOX_CTL	
Uncore SBo 2 PerfMon for SBo 2 Box-Wide Control		Package
Register Address: 735H, 1845	MSR_S2_PMON_EVTNSELO	
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 0		Package
Register Address: 736H, 1846	MSR_S2_PMON_EVTNSEL1	
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 1		Package
Register Address: 737H, 1847	MSR_S2_PMON_EVTNSEL2	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 2		Package
Register Address: 738H, 1848	MSR_S2_PMON_EVNTSEL3	
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 3		Package
Register Address: 739H, 1849	MSR_S2_PMON_BOX_FILTER	
Uncore SBo 2 PerfMon Box-Wide Filter		Package
Register Address: 73AH, 1850	MSR_S2_PMON_CTR0	
Uncore SBo 2 PerfMon Counter 0		Package
Register Address: 73BH, 1851	MSR_S2_PMON_CTR1	
Uncore SBo 2 PerfMon Counter 1		Package
Register Address: 73CH, 1852	MSR_S2_PMON_CTR2	
Uncore SBo 2 PerfMon Counter 2		Package
Register Address: 73DH, 1853	MSR_S2_PMON_CTR3	
Uncore SBo 2 PerfMon Counter 3		Package
Register Address: 73EH, 1854	MSR_S3_PMON_BOX_CTL	
Uncore SBo 3 PerfMon for SBo 3 Box-Wide Control		Package
Register Address: 73FH, 1855	MSR_S3_PMON_EVNTSELO	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 0		Package
Register Address: 740H, 1856	MSR_S3_PMON_EVNTSEL1	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 1		Package
Register Address: 741H, 1857	MSR_S3_PMON_EVNTSEL2	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 2		Package
Register Address: 742H, 1858	MSR_S3_PMON_EVNTSEL3	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 3		Package
Register Address: 743H, 1859	MSR_S3_PMON_BOX_FILTER	
Uncore SBo 3 PerfMon Box-Wide Filter		Package
Register Address: 744H, 1860	MSR_S3_PMON_CTR0	
Uncore SBo 3 PerfMon Counter 0		Package
Register Address: 745H, 1861	MSR_S3_PMON_CTR1	
Uncore SBo 3 PerfMon Counter 1		Package
Register Address: 746H, 1862	MSR_S3_PMON_CTR2	
Uncore SBo 3 PerfMon Counter 2		Package
Register Address: 747H, 1863	MSR_S3_PMON_CTR3	
Uncore SBo 3 PerfMon Counter 3		Package
Register Address: E00H, 3584	MSR_CO_PMON_BOX_CTL	
Uncore C-Box 0 PerfMon for Box-Wide Control		Package
Register Address: E01H, 3585	MSR_CO_PMON_EVNTSELO	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 0		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E02H, 3586	MSR_CO_PMON_EVTSEL1	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 1		Package
Register Address: E03H, 3587	MSR_CO_PMON_EVTSEL2	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 2		Package
Register Address: E04H, 3588	MSR_CO_PMON_EVTSEL3	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 3		Package
Register Address: E05H, 3589	MSR_CO_PMON_BOX_FILTER0	
Uncore C-Box 0 PerfMon Box Wide Filter 0		Package
Register Address: E06H, 3590	MSR_CO_PMON_BOX_FILTER1	
Uncore C-Box 0 PerfMon Box Wide Filter 1		Package
Register Address: E07H, 3591	MSR_CO_PMON_BOX_STATUS	
Uncore C-Box 0 PerfMon Box Wide Status		Package
Register Address: E08H, 3592	MSR_CO_PMON_CTR0	
Uncore C-Box 0 PerfMon Counter 0		Package
Register Address: E09H, 3593	MSR_CO_PMON_CTR1	
Uncore C-Box 0 PerfMon Counter 1		Package
Register Address: E0AH, 3594	MSR_CO_PMON_CTR2	
Uncore C-Box 0 PerfMon Counter 2		Package
Register Address: E0BH, 3595	MSR_CO_PMON_CTR3	
Uncore C-Box 0 PerfMon Counter 3		Package
Register Address: E10H, 3600	MSR_C1_PMON_BOX_CTL	
Uncore C-Box 1 PerfMon for Box-Wide Control		Package
Register Address: E11H, 3601	MSR_C1_PMON_EVTSEL0	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 0		Package
Register Address: E12H, 3602	MSR_C1_PMON_EVTSEL1	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 1		Package
Register Address: E13H, 3603	MSR_C1_PMON_EVTSEL2	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 2		Package
Register Address: E14H, 3604	MSR_C1_PMON_EVTSEL3	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 3		Package
Register Address: E15H, 3605	MSR_C1_PMON_BOX_FILTER0	
Uncore C-Box 1 PerfMon Box Wide Filter 0		Package
Register Address: E16H, 3606	MSR_C1_PMON_BOX_FILTER1	
Uncore C-Box 1 PerfMon Box Wide Filter 1		Package
Register Address: E17H, 3607	MSR_C1_PMON_BOX_STATUS	
Uncore C-Box 1 PerfMon Box Wide Status		Package
Register Address: E18H, 3608	MSR_C1_PMON_CTR0	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 1 PerfMon Counter 0		Package
Register Address: E19H, 3609	MSR_C1_PMON_CTR1	
Uncore C-Box 1 PerfMon Counter 1		Package
Register Address: E1AH, 3610	MSR_C1_PMON_CTR2	
Uncore C-Box 1 PerfMon Counter 2		Package
Register Address: E1BH, 3611	MSR_C1_PMON_CTR3	
Uncore C-Box 1 PerfMon Counter 3		Package
Register Address: E20H, 3616	MSR_C2_PMON_BOX_CTL	
Uncore C-Box 2 PerfMon for Box-Wide Control		Package
Register Address: E21H, 3617	MSR_C2_PMON_EVNTSELO	
Uncore C-Box 2 PerfMon Event Select for C-Box 2 Counter 0		Package
Register Address: E22H, 3618	MSR_C2_PMON_EVNTSEL1	
Uncore C-Box 2 PerfMon Event Select for C-Box 2 Counter 1		Package
Register Address: E23H, 3619	MSR_C2_PMON_EVNTSEL2	
Uncore C-Box 2 PerfMon Event Select for C-Box 2 Counter 2		Package
Register Address: E24H, 3620	MSR_C2_PMON_EVNTSEL3	
Uncore C-Box 2 PerfMon Event select for C-Box 2 Counter 3		Package
Register Address: E25H, 3621	MSR_C2_PMON_BOX_FILTER0	
Uncore C-Box 2 PerfMon Box Wide Filter 0		Package
Register Address: E26H, 3622	MSR_C2_PMON_BOX_FILTER1	
Uncore C-Box 2 PerfMon Box Wide Filter1		Package
Register Address: E27H, 3623	MSR_C2_PMON_BOX_STATUS	
Uncore C-Box 2 PerfMon Box Wide Status		Package
Register Address: E28H, 3624	MSR_C2_PMON_CTR0	
Uncore C-Box 2 PerfMon Counter 0		Package
Register Address: E29H, 3625	MSR_C2_PMON_CTR1	
Uncore C-Box 2 PerfMon Counter 1		Package
Register Address: E2AH, 3626	MSR_C2_PMON_CTR2	
Uncore C-Box 2 PerfMon Counter 2		Package
Register Address: E2BH, 3627	MSR_C2_PMON_CTR3	
Uncore C-Box 2 PerfMon Counter 3		Package
Register Address: E30H, 3632	MSR_C3_PMON_BOX_CTL	
Uncore C-Box 3 PerfMon for Box-Wide Control		Package
Register Address: E31H, 3633	MSR_C3_PMON_EVNTSELO	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 0		Package
Register Address: E32H, 3634	MSR_C3_PMON_EVNTSEL1	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 1		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E33H, 3635	MSR_C3_PMON_EVTSEL2	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 2		Package
Register Address: E34H, 3636	MSR_C3_PMON_EVTSEL3	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 3		Package
Register Address: E35H, 3637	MSR_C3_PMON_BOX_FILTER0	
Uncore C-Box 3 PerfMon Box Wide Filter 0		Package
Register Address: E36H, 3638	MSR_C3_PMON_BOX_FILTER1	
Uncore C-Box 3 PerfMon Box Wide Filter1		Package
Register Address: E37H, 3639	MSR_C3_PMON_BOX_STATUS	
Uncore C-Box 3 PerfMon Box Wide Status		Package
Register Address: E38H, 3640	MSR_C3_PMON_CTR0	
Uncore C-Box 3 PerfMon Counter 0		Package
Register Address: E39H, 3641	MSR_C3_PMON_CTR1	
Uncore C-Box 3 PerfMon Counter 1		Package
Register Address: E3AH, 3642	MSR_C3_PMON_CTR2	
Uncore C-Box 3 PerfMon Counter 2		Package
Register Address: E3BH, 3643	MSR_C3_PMON_CTR3	
Uncore C-Box 3 PerfMon Counter 3		Package
Register Address: E40H, 3648	MSR_C4_PMON_BOX_CTL	
Uncore C-Box 4 PerfMon for Box-Wide Control		Package
Register Address: E41H, 3649	MSR_C4_PMON_EVTSELO	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 0		Package
Register Address: E42H, 3650	MSR_C4_PMON_EVTSEL1	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 1		Package
Register Address: E43H, 3651	MSR_C4_PMON_EVTSEL2	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 2		Package
Register Address: E44H, 3652	MSR_C4_PMON_EVTSEL3	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 3		Package
Register Address: E45H, 3653	MSR_C4_PMON_BOX_FILTER0	
Uncore C-Box 4 PerfMon Box Wide Filter 0		Package
Register Address: E46H, 3654	MSR_C4_PMON_BOX_FILTER1	
Uncore C-Box 4 PerfMon Box Wide Filter1		Package
Register Address: E47H, 3655	MSR_C4_PMON_BOX_STATUS	
Uncore C-Box 4 PerfMon Box Wide Status		Package
Register Address: E48H, 3656	MSR_C4_PMON_CTR0	
Uncore C-Box 4 PerfMon Counter 0		Package
Register Address: E49H, 3657	MSR_C4_PMON_CTR1	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 4 PerfMon Counter 1		Package
Register Address: E4AH, 3658	MSR_C4_PMON_CTR2	
Uncore C-Box 4 PerfMon Counter 2		Package
Register Address: E4BH, 3659	MSR_C4_PMON_CTR3	
Uncore C-Box 4 PerfMon Counter 3		Package
Register Address: E50H, 3664	MSR_C5_PMON_BOX_CTL	
Uncore C-Box 5 PerfMon for Box-Wide Control		Package
Register Address: E51H, 3665	MSR_C5_PMON_EVNTSELO	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 0		Package
Register Address: E52H, 3666	MSR_C5_PMON_EVNTSEL1	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 1		Package
Register Address: E53H, 3667	MSR_C5_PMON_EVNTSEL2	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 2		Package
Register Address: E54H, 3668	MSR_C5_PMON_EVNTSEL3	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 3		Package
Register Address: E55H, 3669	MSR_C5_PMON_BOX_FILTER0	
Uncore C-Box 5 PerfMon Box Wide Filter 0		Package
Register Address: E56H, 3670	MSR_C5_PMON_BOX_FILTER1	
Uncore C-Box 5 PerfMon Box Wide Filter 1		Package
Register Address: E57H, 3671	MSR_C5_PMON_BOX_STATUS	
Uncore C-Box 5 PerfMon Box Wide Status		Package
Register Address: E58H, 3672	MSR_C5_PMON_CTR0	
Uncore C-Box 5 PerfMon Counter 0		Package
Register Address: E59H, 3673	MSR_C5_PMON_CTR1	
Uncore C-Box 5 PerfMon Counter 1		Package
Register Address: E5AH, 3674	MSR_C5_PMON_CTR2	
Uncore C-Box 5 PerfMon Counter 2		Package
Register Address: E5BH, 3675	MSR_C5_PMON_CTR3	
Uncore C-Box 5 PerfMon Counter 3		Package
Register Address: E60H, 3680	MSR_C6_PMON_BOX_CTL	
Uncore C-Box 6 PerfMon for Box-Wide Control		Package
Register Address: E61H, 3681	MSR_C6_PMON_EVNTSELO	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 0		Package
Register Address: E62H, 3682	MSR_C6_PMON_EVNTSEL1	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 1		Package
Register Address: E63H, 3683	MSR_C6_PMON_EVNTSEL2	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 2		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E64H, 3684	MSR_C6_PMON_EVTSEL3	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 3		Package
Register Address: E65H, 3685	MSR_C6_PMON_BOX_FILTER0	
Uncore C-Box 6 PerfMon Box Wide Filter 0		Package
Register Address: E66H, 3686	MSR_C6_PMON_BOX_FILTER1	
Uncore C-Box 6 PerfMon Box Wide Filter 1		Package
Register Address: E67H, 3687	MSR_C6_PMON_BOX_STATUS	
Uncore C-Box 6 PerfMon Box Wide Status		Package
Register Address: E68H, 3688	MSR_C6_PMON_CTR0	
Uncore C-Box 6 PerfMon Counter 0		Package
Register Address: E69H, 3689	MSR_C6_PMON_CTR1	
Uncore C-Box 6 PerfMon Counter 1		Package
Register Address: E6AH, 3690	MSR_C6_PMON_CTR2	
Uncore C-Box 6 PerfMon Counter 2		Package
Register Address: E6BH, 3691	MSR_C6_PMON_CTR3	
Uncore C-Box 6 PerfMon Counter 3		Package
Register Address: E70H, 3696	MSR_C7_PMON_BOX_CTL	
Uncore C-Box 7 PerfMon for Box-Wide Control		Package
Register Address: E71H, 3697	MSR_C7_PMON_EVTSELO	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 0		Package
Register Address: E72H, 3698	MSR_C7_PMON_EVTSEL1	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 1		Package
Register Address: E73H, 3699	MSR_C7_PMON_EVTSEL2	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 2		Package
Register Address: E74H, 3700	MSR_C7_PMON_EVTSEL3	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 3		Package
Register Address: E75H, 3701	MSR_C7_PMON_BOX_FILTER0	
Uncore C-Box 7 PerfMon Box Wide Filter 0		Package
Register Address: E76H, 3702	MSR_C7_PMON_BOX_FILTER1	
Uncore C-Box 7 PerfMon Box Wide Filter 1		Package
Register Address: E77H, 3703	MSR_C7_PMON_BOX_STATUS	
Uncore C-Box 7 PerfMon Box Wide Status		Package
Register Address: E78H, 3704	MSR_C7_PMON_CTR0	
Uncore C-Box 7 PerfMon Counter 0		Package
Register Address: E79H, 3705	MSR_C7_PMON_CTR1	
Uncore C-Box 7 PerfMon Counter 1		Package
Register Address: E7AH, 3706	MSR_C7_PMON_CTR2	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 7 PerfMon Counter 2		Package
Register Address: E7BH, 3707	MSR_C7_PMON_CTR3	
Uncore C-Box 7 PerfMon Counter 3		Package
Register Address: E80H, 3712	MSR_C8_PMON_BOX_CTL	
Uncore C-Box 8 PerfMon Local Box Wide Control		Package
Register Address: E81H, 3713	MSR_C8_PMON_EVNTSELO	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 0		Package
Register Address: E82H, 3714	MSR_C8_PMON_EVNTSEL1	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 1		Package
Register Address: E83H, 3715	MSR_C8_PMON_EVNTSEL2	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 2		Package
Register Address: E84H, 3716	MSR_C8_PMON_EVNTSEL3	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 3		Package
Register Address: E85H, 3717	MSR_C8_PMON_BOX_FILTER0	
Uncore C-Box 8 PerfMon Box Wide Filter 0		Package
Register Address: E86H, 3718	MSR_C8_PMON_BOX_FILTER1	
Uncore C-Box 8 PerfMon Box Wide Filter 1		Package
Register Address: E87H, 3719	MSR_C8_PMON_BOX_STATUS	
Uncore C-Box 8 PerfMon Box Wide Status		Package
Register Address: E88H, 3720	MSR_C8_PMON_CTR0	
Uncore C-Box 8 PerfMon Counter 0		Package
Register Address: E89H, 3721	MSR_C8_PMON_CTR1	
Uncore C-Box 8 PerfMon Counter 1		Package
Register Address: E8AH, 3722	MSR_C8_PMON_CTR2	
Uncore C-Box 8 PerfMon Counter 2		Package
Register Address: E8BH, 3723	MSR_C8_PMON_CTR3	
Uncore C-Box 8 PerfMon Counter 3		Package
Register Address: E90H, 3728	MSR_C9_PMON_BOX_CTL	
Uncore C-Box 9 PerfMon Local Box Wide Control		Package
Register Address: E91H, 3729	MSR_C9_PMON_EVNTSELO	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 0		Package
Register Address: E92H, 3730	MSR_C9_PMON_EVNTSEL1	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 1		Package
Register Address: E93H, 3731	MSR_C9_PMON_EVNTSEL2	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 2		Package
Register Address: E94H, 3732	MSR_C9_PMON_EVNTSEL3	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 3		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E95H, 3733	MSR_C9_PMON_BOX_FILTER0	
Uncore C-Box 9 PerfMon Box Wide Filter 0		Package
Register Address: E96H, 3734	MSR_C9_PMON_BOX_FILTER1	
Uncore C-Box 9 PerfMon Box Wide Filter 1		Package
Register Address: E97H, 3735	MSR_C9_PMON_BOX_STATUS	
Uncore C-Box 9 PerfMon Box Wide Status		Package
Register Address: E98H, 3736	MSR_C9_PMON_CTR0	
Uncore C-Box 9 PerfMon Counter 0		Package
Register Address: E99H, 3737	MSR_C9_PMON_CTR1	
Uncore C-Box 9 PerfMon Counter 1		Package
Register Address: E9AH, 3738	MSR_C9_PMON_CTR2	
Uncore C-Box 9 PerfMon Counter 2		Package
Register Address: E9BH, 3739	MSR_C9_PMON_CTR3	
Uncore C-Box 9 PerfMon Counter 3		Package
Register Address: EA0H, 3744	MSR_C10_PMON_BOX_CTL	
Uncore C-Box 10 PerfMon Local Box Wide Control		Package
Register Address: EA1H, 3745	MSR_C10_PMON_EVTNSEL0	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 0		Package
Register Address: EA2H, 3746	MSR_C10_PMON_EVTNSEL1	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 1		Package
Register Address: EA3H, 3747	MSR_C10_PMON_EVTNSEL2	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 2		Package
Register Address: EA4H, 3748	MSR_C10_PMON_EVTNSEL3	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 3		Package
Register Address: EA5H, 3749	MSR_C10_PMON_BOX_FILTER0	
Uncore C-Box 10 PerfMon Box Wide Filter 0		Package
Register Address: EA6H, 3750	MSR_C10_PMON_BOX_FILTER1	
Uncore C-Box 10 PerfMon Box Wide Filter 1		Package
Register Address: EA7H, 3751	MSR_C10_PMON_BOX_STATUS	
Uncore C-Box 10 PerfMon Box Wide Status		Package
Register Address: EA8H, 3752	MSR_C10_PMON_CTR0	
Uncore C-Box 10 PerfMon Counter 0		Package
Register Address: EA9H, 3753	MSR_C10_PMON_CTR1	
Uncore C-Box 10 PerfMon Counter 1		Package
Register Address: EAAH, 3754	MSR_C10_PMON_CTR2	
Uncore C-Box 10 PerfMon Counter 2		Package
Register Address: EABH, 3755	MSR_C10_PMON_CTR3	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 10 PerfMon Counter 3		Package
Register Address: EB0H, 3760	MSR_C11_PMON_BOX_CTL	
Uncore C-Box 11 PerfMon Local Box Wide Control		Package
Register Address: EB1H, 3761	MSR_C11_PMON_EVTNSELO	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 0		Package
Register Address: EB2H, 3762	MSR_C11_PMON_EVTNSEL1	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 1		Package
Register Address: EB3H, 3763	MSR_C11_PMON_EVTNSEL2	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 2		Package
Register Address: EB4H, 3764	MSR_C11_PMON_EVTNSEL3	
Uncore C-box 11 PerfMon Event Select for C-Box 11 Counter 3		Package
Register Address: EB5H, 3765	MSR_C11_PMON_BOX_FILTER0	
Uncore C-Box 11 PerfMon Box Wide Filter 0		Package
Register Address: EB6H, 3766	MSR_C11_PMON_BOX_FILTER1	
Uncore C-Box 11 PerfMon Box Wide Filter 1		Package
Register Address: EB7H, 3767	MSR_C11_PMON_BOX_STATUS	
Uncore C-Box 11 PerfMon Box Wide Status		Package
Register Address: EB8H, 3768	MSR_C11_PMON_CTR0	
Uncore C-Box 11 PerfMon Counter 0		Package
Register Address: EB9H, 3769	MSR_C11_PMON_CTR1	
Uncore C-Box 11 PerfMon Counter 1		Package
Register Address: EBAH, 3770	MSR_C11_PMON_CTR2	
Uncore C-Box 11 PerfMon Counter 2		Package
Register Address: EBBH, 3771	MSR_C11_PMON_CTR3	
Uncore C-Box 11 PerfMon Counter 3		Package
Register Address: EC0H, 3776	MSR_C12_PMON_BOX_CTL	
Uncore C-Box 12 PerfMon Local Box Wide Control		Package
Register Address: EC1H, 3777	MSR_C12_PMON_EVTNSELO	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 0		Package
Register Address: EC2H, 3778	MSR_C12_PMON_EVTNSEL1	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 1		Package
Register Address: EC3H, 3779	MSR_C12_PMON_EVTNSEL2	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 2		Package
Register Address: EC4H, 3780	MSR_C12_PMON_EVTNSEL3	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 3		Package
Register Address: EC5H, 3781	MSR_C12_PMON_BOX_FILTER0	
Uncore C-Box 12 PerfMon Box Wide Filter 0		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: EC6H, 3782	MSR_C12_PMON_BOX_FILTER1	
Uncore C-Box 12 PerfMon Box Wide Filter 1		Package
Register Address: EC7H, 3783	MSR_C12_PMON_BOX_STATUS	
Uncore C-Box 12 PerfMon Box Wide Status		Package
Register Address: EC8H, 3784	MSR_C12_PMON_CTR0	
Uncore C-Box 12 PerfMon Counter 0		Package
Register Address: EC9H, 3785	MSR_C12_PMON_CTR1	
Uncore C-Box 12 PerfMon Counter 1		Package
Register Address: ECAH, 3786	MSR_C12_PMON_CTR2	
Uncore C-Box 12 PerfMon Counter 2		Package
Register Address: ECBH, 3787	MSR_C12_PMON_CTR3	
Uncore C-Box 12 PerfMon Counter 3		Package
Register Address: EDOH, 3792	MSR_C13_PMON_BOX_CTL	
Uncore C-Box 13 PerfMon local box wide control.		Package
Register Address: ED1H, 3793	MSR_C13_PMON_EVTSELO	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 0		Package
Register Address: ED2H, 3794	MSR_C13_PMON_EVTSEL1	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 1		Package
Register Address: ED3H, 3795	MSR_C13_PMON_EVTSEL2	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 2		Package
Register Address: ED4H, 3796	MSR_C13_PMON_EVTSEL3	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 3		Package
Register Address: ED5H, 3797	MSR_C13_PMON_BOX_FILTER0	
Uncore C-Box 13 PerfMon Box Wide Filter 0		Package
Register Address: ED6H, 3798	MSR_C13_PMON_BOX_FILTER1	
Uncore C-Box 13 PerfMon Box Wide Filter 1		Package
Register Address: ED7H, 3799	MSR_C13_PMON_BOX_STATUS	
Uncore C-Box 13 PerfMon Box Wide Status		Package
Register Address: ED8H, 3800	MSR_C13_PMON_CTR0	
Uncore C-Box 13 PerfMon Counter 0		Package
Register Address: ED9H, 3801	MSR_C13_PMON_CTR1	
Uncore C-Box 13 PerfMon Counter 1		Package
Register Address: EDAH, 3802	MSR_C13_PMON_CTR2	
Uncore C-Box 13 PerfMon Counter 2		Package
Register Address: EDBH, 3803	MSR_C13_PMON_CTR3	
Uncore C-Box 13 PerfMon Counter 3		Package
Register Address: EEOH, 3808	MSR_C14_PMON_BOX_CTL	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 14 PerfMon Local Box Wide Control		Package
Register Address: EE1H, 3809	MSR_C14_PMON_EVNTSEL0	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 0		Package
Register Address: EE2H, 3810	MSR_C14_PMON_EVNTSEL1	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 1		Package
Register Address: EE3H, 3811	MSR_C14_PMON_EVNTSEL2	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 2		Package
Register Address: EE4H, 3812	MSR_C14_PMON_EVNTSEL3	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 3		Package
Register Address: EE5H, 3813	MSR_C14_PMON_BOX_FILTER	
Uncore C-Box 14 PerfMon Box Wide Filter 0		Package
Register Address: EE6H, 3814	MSR_C14_PMON_BOX_FILTER1	
Uncore C-Box 14 PerfMon Box Wide Filter 1		Package
Register Address: EE7H, 3815	MSR_C14_PMON_BOX_STATUS	
Uncore C-Box 14 PerfMon Box Wide Status		Package
Register Address: EE8H, 3816	MSR_C14_PMON_CTR0	
Uncore C-Box 14 PerfMon Counter 0		Package
Register Address: EE9H, 3817	MSR_C14_PMON_CTR1	
Uncore C-Box 14 PerfMon Counter 1		Package
Register Address: EEAH, 3818	MSR_C14_PMON_CTR2	
Uncore C-Box 14 PerfMon Counter 2		Package
Register Address: EEBH, 3819	MSR_C14_PMON_CTR3	
Uncore C-Box 14 PerfMon Counter 3		Package
Register Address: EF0H, 3824	MSR_C15_PMON_BOX_CTL	
Uncore C-Box 15 PerfMon Local Box Wide Control		Package
Register Address: EF1H, 3825	MSR_C15_PMON_EVNTSEL0	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 0		Package
Register Address: EF2H, 3826	MSR_C15_PMON_EVNTSEL1	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 1		Package
Register Address: EF3H, 3827	MSR_C15_PMON_EVNTSEL2	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 2		Package
Register Address: EF4H, 3828	MSR_C15_PMON_EVNTSEL3	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 3		Package
Register Address: EF5H, 3829	MSR_C15_PMON_BOX_FILTER0	
Uncore C-Box 15 PerfMon Box Wide Filter 0		Package
Register Address: EF6H, 3830	MSR_C15_PMON_BOX_FILTER1	
Uncore C-Box 15 PerfMon Box Wide Filter 1		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: EF7H, 3831	MSR_C15_PMON_BOX_STATUS	
Uncore C-Box 15 PerfMon Box Wide Status		Package
Register Address: EF8H, 3832	MSR_C15_PMON_CTR0	
Uncore C-Box 15 PerfMon Counter 0		Package
Register Address: EF9H, 3833	MSR_C15_PMON_CTR1	
Uncore C-Box 15 PerfMon Counter 1		Package
Register Address: EFAH, 3834	MSR_C15_PMON_CTR2	
Uncore C-Box 15 PerfMon Counter 2		Package
Register Address: EFBH, 3835	MSR_C15_PMON_CTR3	
Uncore C-Box 15 PerfMon Counter 3		Package
Register Address: F00H, 3840	MSR_C16_PMON_BOX_CTL	
Uncore C-Box 16 PerfMon for Box-Wide Control		Package
Register Address: F01H, 3841	MSR_C16_PMON_EVTSELO	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 0		Package
Register Address: F02H, 3842	MSR_C16_PMON_EVTSEL1	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 1		Package
Register Address: F03H, 3843	MSR_C16_PMON_EVTSEL2	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 2		Package
Register Address: F04H, 3844	MSR_C16_PMON_EVTSEL3	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 3		Package
Register Address: F05H, 3845	MSR_C16_PMON_BOX_FILTER0	
Uncore C-Box 16 PerfMon Box Wide Filter 0		Package
Register Address: F06H, 3846	MSR_C16_PMON_BOX_FILTER1	
Uncore C-Box 16 PerfMon Box Wide Filter 1		Package
Register Address: F07H, 3847	MSR_C16_PMON_BOX_STATUS	
Uncore C-Box 16 PerfMon Box Wide Status		Package
Register Address: F08H, 3848	MSR_C16_PMON_CTR0	
Uncore C-Box 16 PerfMon Counter 0		Package
Register Address: F09H, 3849	MSR_C16_PMON_CTR1	
Uncore C-Box 16 PerfMon Counter 1		Package
Register Address: FOAH, 3850	MSR_C16_PMON_CTR2	
Uncore C-Box 16 PerfMon Counter 2		Package
Register Address: F0BH, 3851	MSR_C16_PMON_CTR3	
Uncore C-Box 16 PerfMon Counter 3		Package
Register Address: F10H, 3856	MSR_C17_PMON_BOX_CTL	
Uncore C-Box 17 PerfMon for Box-Wide Control		Package
Register Address: F11H, 3857	MSR_C17_PMON_EVTSELO	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 0		Package
Register Address: F12H, 3858	MSR_C17_PMON_EVTNSEL1	
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 1		Package
Register Address: F13H, 3859	MSR_C17_PMON_EVTNSEL2	
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 2		Package
Register Address: F14H, 3860	MSR_C17_PMON_EVTNSEL3	
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 3		Package
Register Address: F15H, 3861	MSR_C17_PMON_BOX_FILTER0	
Uncore C-Box 17 PerfMon Box Wide Filter 0		Package
Register Address: F16H, 3862	MSR_C17_PMON_BOX_FILTER1	
Uncore C-Box 17 PerfMon Box Wide Filter 1		Package
Register Address: F17H, 3863	MSR_C17_PMON_BOX_STATUS	
Uncore C-Box 17 PerfMon Box Wide Status		Package
Register Address: F18H, 3864	MSR_C17_PMON_CTR0	
Uncore C-Box 17 PerfMon Counter 0		Package
Register Address: F19H, 3865	MSR_C17_PMON_CTR1	
Uncore C-Box 17 PerfMon Counter 1		Package
Register Address: F1AH, 3866	MSR_C17_PMON_CTR2	
Uncore C-Box 17 PerfMon Counter 2		Package
Register Address: F1BH, 3867	MSR_C17_PMON_CTR3	
Uncore C-Box 17 PerfMon Counter 3		Package

2.15 MSRS IN THE INTEL® CORE™ M PROCESSORS AND THE 5TH GENERATION INTEL® CORE™ PROCESSORS

The Intel® Core™ M-5xxx processors, 5th generation Intel® Core™ Processors, and the Intel® Xeon® Processor E3-1200 v4 family are based on Broadwell microarchitecture. The Intel® Core™ M-5xxx processors and 5th generation Intel® Core™ Processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH. The Intel® Xeon® Processor E3-1200 v4 family and 5th generation Intel® Core™ Processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_47H. Processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH or 06_47H support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29, Table 2-30, Table 2-34, and Table 2-35. For an MSR listed in Table 2-35 that also appears in the model-specific tables of prior generations, Table 2-35 supersedes prior generation tables.

Table 2-34 lists MSRs that are common to processors based on the Broadwell microarchitectures (including CPUID Signature DisplayFamily_DisplayModel values of 06_3DH, 06_47H, 06_4FH, and 06_56H).

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2 and Section 22.6.2.2, "Global Counter Control Facilities."		Thread
0	Ovf_PMC0	
1	Ovf_PMC1	
2	Ovf_PMC2	
3	Ovf_PMC3	
31:4	Reserved	
32	Ovf_FixedCtr0	
33	Ovf_FixedCtr1	
34	Ovf_FixedCtr2	
54:35	Reserved.	
55	Trace_ToPA_PMI See Section 36.2.7.2, "Table of Physical Addresses (ToPA)."	
60:56	Reserved.	
61	Ovf_Uncore	
62	Ovf_BufDSSAVE	
63	CondChgd	
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2 and Section 22.6.2.2, "Global Counter Control Facilities."		Thread
0	Set 1 to clear Ovf_PMC0.	
1	Set 1 to clear Ovf_PMC1.	
2	Set 1 to clear Ovf_PMC2.	
3	Set 1 to clear Ovf_PMC3.	
31:4	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	
33	Set 1 to clear Ovf_FixedCtr1.	
34	Set 1 to clear Ovf_FixedCtr2	
54:35	Reserved.	
55	Set 1 to clear Trace_ToPA_PMI. See Section 36.2.7.2, "Table of Physical Addresses (ToPA)."	
60:56	Reserved.	
61	Set 1 to clear Ovf_Uncore.	
62	Set 1 to clear Ovf_BufDSSAVE.	
63	Set 1 to clear CondChgd.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W)		Thread
6:0	Reserved.	
M-1:7	Base physical address. M is the value enumerated by CPUID.80000008H:EAX[7:0].	

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:M	Reserved.	
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W)		Thread
6:0	Reserved.	
31:7	MaskOrTableOffset	
63:32	Output Offset.	
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Thread
0	TraceEn	
1	Reserved, must be zero.	
2	OS	
3	User	
6:4	Reserved, must be zero.	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	Reserved, must be zero.	
10	TSCEn	
11	DisRETC	
12	Reserved, must be zero.	
13	Reserved; writing 0 will #GP if also setting TraceEn.	
63:14	Reserved, must be zero.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Thread
0	Reserved, writes ignored.	
1	ContexEn, writes ignored.	
2	TriggerEn, writes ignored.	
3	Reserved	
4	Error (R/W)	
5	Stopped	
63:6	Reserved, must be zero.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Thread
4:0	Reserved.	
63:5	CR3[63:5] value to match.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	

Table 2-35 lists MSRs that are specific to Intel Core M processors and 5th Generation Intel Core Processors.

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10	
9:4	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/W0)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Enable Package C-State Auto-Demotion (R/W)	
30	Enable Package C-State Undemotion (R/W)	
63:31	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6core active.	Package
63:48	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
See Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29, Table 2-30, and Table 2-34 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH.		

2.16 MSRS IN THE INTEL® XEON® PROCESSOR E5 V4 FAMILY

The MSRs listed in Table 2-36 are available and common to the Intel® Xeon® Processor D Product Family (CPUID Signature DisplayFamily_DisplayModel value of 06_56H) and to the Intel Xeon processors E5 v4 and E7 v4 families (CPUID Signature DisplayFamily_DisplayModel value of 06_4FH). These processors are based on Broadwell microarchitecture.

See Section 2.16.1 for lists of tables of MSRs that are supported by the Intel® Xeon® Processor D Family.

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Protected Processor Inventory Number	Enable Control (R/W)	Package
0	LockOut (R/WO) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number	(R/O)	Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information		Package
Contains power management and other model specific features enumeration. See http://biosbits.org .		
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) See Table 2-26.	Package
22:16	Reserved.	
23	PPIN_CAP (R/O) See Table 2-26.	Package
27:24	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.	Package
30	Programmable TJ OFFSET (R/O) See Table 2-26.	Package
39:31	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) See Table 2-26.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W)		Core
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/WO)	
16	Automatic C-State Conversion Enable (R/W) If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6).	
24:17	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	MCG_EM_P	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.	
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.	
63:60	Reserved.	
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	
7	Thermal Threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WC0) See Table 2-2.	
12	Current Limit Status (R/O) See Table 2-2.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Current Limit Log (R/WC0) See Table 2-2.	
14	Cross Domain Limit Status (R/O) See Table 2-2.	
15	Cross Domain Limit Log (R/WC0) See Table 2-2.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R/O) See Table 2-26.	
27:24	TCC Activation Offset (R/W) See Table 2-26.	
63:28	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C	Package
15:8	Maximum Ratio Limit for 2C	Package
23:16	Maximum Ratio Limit for 3C	Package
31:24	Maximum Ratio Limit for 4C	Package
39:32	Maximum Ratio Limit for 5C	Package
47:40	Maximum Ratio Limit for 6C	Package
55:48	Maximum Ratio Limit for 7C	Package
63:56	Maximum Ratio Limit for 8C	Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C	Package
15:8	Maximum Ratio Limit for 10C	Package

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
23:16	Maximum Ratio Limit for 11C	Package
31:24	Maximum Ratio Limit for 12C	Package
39:32	Maximum Ratio Limit for 13C	Package
47:40	Maximum Ratio Limit for 14C	Package
55:48	Maximum Ratio Limit for 15C	Package
63:56	Maximum Ratio Limit for 16C	Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 17.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 17.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
63:15	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
7	Reserved.	
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
2	Power Budget Management Status (R0) When set, frequency is reduced below the operating system request due to PBM limit.	
3	Platform Configuration Services Status (R0) When set, frequency is reduced below the operating system request due to PCS limit.	
4	Reserved.	
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Reserved.	
10	Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits.	
12:11	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1.	
14	Core Max N-Core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency.	
15	Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
18	Power Budget Management Log When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19	Platform Configuration Services Log When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20	Reserved.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Multi-Core Turbo Log When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28:27	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
29	Core Frequency P1 Log When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
30	Core Max N-Core Turbo Frequency Limiting Log When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
31	Core Frequency Limiting Log When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:32	Reserved.	
Register Address: 770H, 1904	IA32_PM_ENABLE	
See Section 17.4.2, "Enabling HWP."		Package
Register Address: 771H, 1905	IA32_HWP_CAPABILITIES	
See Section 17.4.3, "HWP Performance Range and Dynamic Capabilities."		Thread
Register Address: 774H, 1908	IA32_HWP_REQUEST	
See Section 17.4.4, "Managing HWP."		Thread
7:0	Minimum Performance (R/W)	
15:8	Maximum Performance (R/W)	
23:16	Desired Performance (R/W)	
63:24	Reserved.	
Register Address: 777H, 1911	IA32_HWP_STATUS	
See Section 17.4.5, "HWP Feedback."		Thread
1:0	Reserved.	
2	Excursion to Minimum (R/O)	
63:3	Reserved.	
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread
7:0	EventID (R/W) Event encoding: 0x00: No monitoring. 0x01: L3 occupancy monitoring. 0x02: Total memory bandwidth monitoring. 0x03: Local memory bandwidth monitoring. All other encoding reserved.	
31:8	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
41:32	RMID (R/W)	
63:42	Reserved.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
9:0	RMID	
31:10	Reserved.	
51:32	CLOS (R/W)	
63: 52	Reserved.	
Register Address: C90H, 3216	IA32_L3_QOS_MASK_0	
L3 Class Of Service Mask - CLOS 0 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 0.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 0 enforcement.	
63:20	Reserved.	
Register Address: C91H, 3217	IA32_L3_QOS_MASK_1	
L3 Class Of Service Mask - CLOS 1 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 1.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 1 enforcement.	
63:20	Reserved.	
Register Address: C92H, 3218	IA32_L3_QOS_MASK_2	
L3 Class Of Service Mask - CLOS 2 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 2.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 2 enforcement.	
63:20	Reserved.	
Register Address: C93H, 3219	IA32_L3_QOS_MASK_3	
L3 Class Of Service Mask - CLOS 3 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 3.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: C94H, 3220	IA32_L3_QOS_MASK_4	
L3 Class Of Service Mask - CLOS 4 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 4.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 4 enforcement.	
63:20	Reserved.	
Register Address: C95H, 3221	IA32_L3_QOS_MASK_5	
L3 Class Of Service Mask - CLOS 5 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 5.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 5 enforcement.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:20	Reserved.	
Register Address: C96H, 3222	IA32_L3_QOS_MASK_6	
L3 Class Of Service Mask - CLOS 6 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 6.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 6 enforcement.	
63:20	Reserved.	
Register Address: C97H, 3223	IA32_L3_QOS_MASK_7	
L3 Class Of Service Mask - CLOS 7 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 7.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 7 enforcement.	
63:20	Reserved.	
Register Address: C98H, 3224	IA32_L3_QOS_MASK_8	
L3 Class Of Service Mask - CLOS 8 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 8.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 8 enforcement.	
63:20	Reserved.	
Register Address: C99H, 3225	IA32_L3_QOS_MASK_9	
L3 Class Of Service Mask - CLOS 9 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 9.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 9 enforcement.	
63:20	Reserved.	
Register Address: C9AH, 3226	IA32_L3_QOS_MASK_10	
L3 Class Of Service Mask - CLOS 10 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 10.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 10 enforcement.	
63:20	Reserved.	
Register Address: C9BH, 3227	IA32_L3_QOS_MASK_11	
L3 Class Of Service Mask - CLOS 11 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 11.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 11 enforcement.	
63:20	Reserved.	
Register Address: C9CH, 3228	IA32_L3_QOS_MASK_12	
L3 Class Of Service Mask - CLOS 12 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 12.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 12 enforcement.	
63:20	Reserved.	
Register Address: C9DH, 3229	IA32_L3_QOS_MASK_13	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
L3 Class Of Service Mask - CLOS 13 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 13.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 13 enforcement.	
63:20	Reserved.	
Register Address: C9EH, 3230	IA32_L3_QOS_MASK_14	
L3 Class Of Service Mask - CLOS 14 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 14.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 14 enforcement.	
63:20	Reserved.	
Register Address: C9FH, 3231	IA32_L3_QOS_MASK_15	
L3 Class Of Service Mask - CLOS 15 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 15.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 15 enforcement.	
63:20	Reserved.	

2.16.1 Additional MSRs Supported in the Intel® Xeon® Processor D Product Family

The MSRs listed in Table 2-37 are available to Intel® Xeon® Processor D Product Family (CPUID Signature DisplayFamily_DisplayModel value of 06_56H). The Intel® Xeon® processor D product family is based on Broadwell microarchitecture and supports the MSR interfaces listed in Table 2-20, Table 2-29, Table 2-34, Table 2-36, and Table 2-37.

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ACH, 428	MSR_TURBO_RATIO_LIMIT3	
Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
62:0	Reserved.	Package
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	
Register Address: 449H, 1097	IA32_MC18_STATUS	

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
See Table 2-20, Table 2-29, Table 2-34, and Table 2-36 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_56H.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.16.2 Additional MSRs Supported in Intel® Xeon® Processors E5 v4 and E7 v4 Families

The MSRs listed in Table 2-37 are available to the Intel® Xeon® Processor E5 v4 and E7 v4 Families (CPUID Signature DisplayFamily_DisplayModel value of 06_4FH). The Intel® Xeon® processor E5 v4 family is based on Broadwell microarchitecture and supports the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-29, Table 2-34, Table 2-36, and Table 2-38.

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ACH, 428	MSR_TURBO_RATIO_LIMIT3	
Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
62:0	Reserved.	Package
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1, and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.	Package	
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.	Package	
Register Address: 44EH, 1102	IA32_MC19_ADDR	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 454H, 1108	IA32_MC21_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: C81H, 3201	IA32_L3_QOS_CFG	
Cache Allocation Technology Configuration (R/W)		Package
0	CAT Enable. Set 1 to enable Cache Allocation Technology.	
63:1	Reserved.	
See Table 2-20, Table 2-21, Table 2-29, and Table 2-30 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.17 MSRS IN THE 6TH—13TH GENERATION INTEL® CORE™ PROCESSORS, 1ST—5TH GENERATION INTEL® XEON® SCALABLE PROCESSOR FAMILIES, INTEL® CORE™ ULTRA 7 PROCESSORS, 8TH GENERATION INTEL® CORE™ I3 PROCESSORS, INTEL® XEON® E PROCESSORS, INTEL® XEON® 6 P-CORE PROCESSORS, INTEL® XEON® 6 E-CORE PROCESSORS, AND INTEL® SERIES 2 CORE™ ULTRA PROCESSORS

6th generation Intel® Core™ processors are based on Skylake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_4EH or 06_5EH.

The Intel® Xeon® Scalable Processor Family based on the Skylake microarchitecture, the 2nd generation Intel® Xeon® Scalable Processor Family based on the Cascade Lake product, and the 3rd generation Intel® Xeon® Scalable Processor Family based on the Cooper Lake product all have a CPUID Signature DisplayFamily_DisplayModel value of 06_55H.

7th generation Intel® Core™ processors are based on the Kaby Lake microarchitecture, 8th generation and 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on Coffee Lake microarchitecture; these processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_8EH or 06_9EH.

8th generation Intel® Core™ i3 processors are based on Cannon Lake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_66H.

10th generation Intel® Core™ processors are based on Comet Lake microarchitecture (with a CPUID Signature DisplayFamily_DisplayModel value of 06_A5H or 06_A6H) and Ice Lake microarchitecture (with a CPUID Signature DisplayFamily_DisplayModel value of 06_7EH).

11th generation Intel® Core™ processors are based on Tiger Lake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_8CH or 06_8DH.

The 3rd generation Intel® Xeon® Scalable Processor Family is based on Ice Lake microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_6AH or 06_6CH.

12th generation Intel® Core™ processors supporting the Alder Lake performance hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_97H or 06_9AH.

13th generation Intel® Core™ processors supporting the Raptor Lake performance hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_BAH, 06_B7H, or 06_BFH.

The 4th generation Intel® Xeon® Scalable Processor Family is based on Sapphire Rapids microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_8FH.

The 5th generation Intel® Xeon® Scalable Processor Family is based on Emerald Rapids microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_CFH.

The Intel® Core™ Ultra 7 processors supporting the Meteor Lake hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_AAH.

The Intel® Xeon® 6 P-core processor is based on the Granite Rapids microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_ADH or 06_AEH.

The Intel® Xeon® 6 E-core processor is based on the Sierra Forest microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_AFH.

The Intel® Series 2 Core™ Ultra processors supporting the Lunar Lake performance hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_BDH.

These processors support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-25, Table 2-29, Table 2-35, and Table 2-39¹. For an MSR listed in Table 2-39 that also appears in the model-specific tables of prior generations, Table 2-39 supersedes prior generation tables.

Tables 2-40 through 2-60 list additional supported MSR interfaces introduced in specific processors; see each table for additional details.

The notation of “Platform” in the Scope column (with respect to MSR_PLATFORM_ENERGY_COUNTER and MSR_PLATFORM_POWER_LIMIT) is limited to the power-delivery domain and the specifics of the power delivery integration may vary by platform vendor’s implementation.

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
MTRR Capability (R/O, Architectural) See Table 2-2		Thread
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal threshold #1 Status (R/O) See Table 2-2.	
7	Thermal threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	

1. MSRs at the following addresses are not supported in the 12th generation Intel Core processor E-core: 3F7H. MSRs at the following addresses are not supported in the 12th generation Intel Core processor E-core or P-core: 652H, 653H, 655H, 656H, DB0H, DB1H, DB2H, and D90H.

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WC0) See Table 2-2.	
12	Current Limit Status (R/O) See Table 2-2.	
13	Current Limit Log (R/WC0) See Table 2-2.	
14	Cross Domain Limit Status (R/O) See Table 2-2.	
15	Cross Domain Limit Log (R/WC0) See Table 2-2.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record.		Thread

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org .		Core
0	Reserved.	
1	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	Package
18:2	Reserved.	
19	Disable Energy Efficiency Optimization (R/W) Setting this bit disables the P-States energy efficiency optimization. Default value is 0. Disable/enable the energy efficiency optimization in P-State legacy mode (when IA32_PM_ENABLE[HWP_ENABLE] = 0), has an effect only in the turbo range or into PERF_MIN_CTL value if it is not zero set. In HWP mode (IA32_PM_ENABLE[HWP_ENABLE] == 1), has an effect between the OS desired or OS maximize to the OS minimize performance setting.	
20	Disable Race to Halt Optimization (R/W) Setting this bit disables the Race to Halt optimization and avoids this optimization limitation to execute below the most efficient frequency ratio. Default value is 0 for processors that support Race to Halt optimization.	
63:21	Reserved.	
Register Address: 300H, 768	MSR_SGXOWNEREPOCH0	
Lower 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 301H, 769	MSR_SGXOWNEREPOCH1	
Upper 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 22.2.4, "Architectural Performance Monitoring Version 4."		
0	Ovf_PMC0	Thread
1	Ovf_PMC1	Thread
2	Ovf_PMC2	Thread
3	Ovf_PMC3	Thread
4	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)	Thread
5	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)	Thread

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
6	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)	Thread
7	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)	Thread
31:8	Reserved.	
32	Ovf_FixedCtr0	Thread
33	Ovf_FixedCtr1	Thread
34	Ovf_FixedCtr2	Thread
54:35	Reserved	
55	Trace_ToPA_PMI	Thread
57:56	Reserved.	
58	LBR_Frz	Thread
59	CTR_Frz	Thread
60	ASCI	Thread
61	Ovf_Uncore	Thread
62	Ovf_BufDSSAVE	Thread
63	CondChgd	Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_STATUS_RESET	
See Table 2-2 and Section 22.2.4, “Architectural Performance Monitoring Version 4.”		
0	Set 1 to clear Ovf_PMC0.	Thread
1	Set 1 to clear Ovf_PMC1.	Thread
2	Set 1 to clear Ovf_PMC2.	Thread
3	Set 1 to clear Ovf_PMC3.	Thread
4	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).	Thread
5	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).	Thread
6	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).	Thread
7	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).	Thread
31:8	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	Thread
33	Set 1 to clear Ovf_FixedCtr1.	Thread
34	Set 1 to clear Ovf_FixedCtr2.	Thread
54:35	Reserved.	
55	Set 1 to clear Trace_ToPA_PMI.	Thread
57:56	Reserved.	
58	Set 1 to clear LBR_Frz.	Thread
59	Set 1 to clear CTR_Frz.	Thread
60	Set 1 to clear ASCI.	Thread

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
61	Set 1 to clear Ovf_Uncore.	Thread
62	Set 1 to clear Ovf_BufDSSAVE.	Thread
63	Set 1 to clear CondChgd.	Thread
Register Address: 391H, 913	IA32_PERF_GLOBAL_STATUS_SET	
See Table 2-2 and Section 22.2.4, “Architectural Performance Monitoring Version 4.”		
0	Set 1 to cause Ovf_PMC0 = 1.	Thread
1	Set 1 to cause Ovf_PMC1 = 1.	Thread
2	Set 1 to cause Ovf_PMC2 = 1.	Thread
3	Set 1 to cause Ovf_PMC3 = 1.	Thread
4	Set 1 to cause Ovf_PMC4 =1 (if CPUID.0AH:EAX[15:8] > 4).	Thread
5	Set 1 to cause Ovf_PMC5 =1 (if CPUID.0AH:EAX[15:8] > 5).	Thread
6	Set 1 to cause Ovf_PMC6 =1 (if CPUID.0AH:EAX[15:8] > 6).	Thread
7	Set 1 to cause Ovf_PMC7 =1 (if CPUID.0AH:EAX[15:8] > 7).	Thread
31:8	Reserved.	
32	Set 1 to cause Ovf_FixedCtr0 = 1.	Thread
33	Set 1 to cause Ovf_FixedCtr1 = 1.	Thread
34	Set 1 to cause Ovf_FixedCtr2 = 1.	Thread
54:35	Reserved.	
55	Set 1 to cause Trace_ToPA_PMI = 1.	Thread
57:56	Reserved.	
58	Set 1 to cause LBR_Frz = 1.	Thread
59	Set 1 to cause CTR_Frz = 1.	Thread
60	Set 1 to cause ASCI = 1.	Thread
61	Set 1 to cause Ovf_Uncore.	Thread
62	Set 1 to cause Ovf_BufDSSAVE.	Thread
63	Reserved.	
Register Address: 392H, 914	IA32_PERF_GLOBAL_INUSE	
See Table 2-2.		Thread
Register Address: 3F7H, 1015	MSR_PEBS_FRONTEND	
FrontEnd Precise Event Condition Select (R/W)		Thread
2:0	Event Code Select	
3	Reserved	
4	Event Code Select High	
7:5	Reserved.	
19:8	IDQ_Bubble_Length Specifier	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
22:20	IDQ_Bubble_Width Specifier	
63:23	Reserved.	
Register Address: 500H, 1280	IA32_SGX_SVN_STATUS	
Status and SVN Threshold of SGX Support for ACM (R/O)		Thread
0	Lock See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."	
15:1	Reserved.	
23:16	SGX_SVN_SINIT See Section 42.11.3, "Interactions with Authenticated Code Modules (ACMs)."	
63:24	Reserved.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W) See Table 2-2.		Thread
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W) See Table 2-2.		Thread
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Thread
0	TraceEn	
1	CYCEn	
2	OS	
3	User	
6:4	Reserved, must be zero.	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	MTCEn	
10	TSCEn	
11	DisRETC	
12	Reserved, must be zero.	
13	BranchEn	
17:14	MTCFreq	
18	Reserved, must be zero.	
22:19	CycThresh	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
23	Reserved, must be zero.	
27:24	PSBFreq	
31:28	Reserved, must be zero.	
35:32	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Thread
0	FilterEn, writes ignored.	
1	ContexEn, writes ignored.	
2	TriggerEn, writes ignored.	
3	Reserved	
4	Error (R/W)	
5	Stopped	
31:6	Reserved, must be zero.	
48:32	PacketByteCnt	
63:49	Reserved, must be zero.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Thread
4:0	Reserved	
63:5	CR3[63:5] value to match	
Register Address: 580H, 1408	IA32_RTIT_ADDR0_A	
Region 0 Start Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 581H, 1409	IA32_RTIT_ADDR0_B	
Region 0 End Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 582H, 1410	IA32_RTIT_ADDR1_A	
Region 1 Start Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 583H, 1411	IA32_RTIT_ADDR1_B	
Region 1 End Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
PP0 Energy Status (R/O) See Section 17.10.4, “PP0/PP1 RAPL Domains.”		Package
Register Address: 64DH, 1613	MSR_PLATFORM_ENERGY_COUNTER	
Platform Energy Counter (R/O) This MSR is valid only if both platform vendor hardware implementation and BIOS enablement support it. This MSR will read 0 if not valid.		Platform
31:0	Total energy consumed by all devices in the platform that receive power from integrated power delivery mechanism, included platform devices are processor cores, SOC, memory, add-on or peripheral devices that get powered directly from the platform power delivery means. The energy units are specified in the MSR_RAPL_POWER_UNIT.Energy_Status_Unit.	
63:32	Reserved.	
Register Address: 64EH, 1614	MSR_PPERF	
Productive Performance Count (R/O)		Thread
63:0	Hardware’s view of workload scalability. See Section 17.4.5.1.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)	Package
0	PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	Residency State Regulation Status (R0) When set, frequency is reduced below the operating system request due to residency state regulation limit.	
5	Running Average Thermal Limit Status (R0) When set, frequency is reduced below the operating system request due to Running Average Thermal Limit (RATL).	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from a processor Voltage Regulator (VR).	
7	VR Therm Design Current Status (R0) When set, frequency is reduced below the operating system request due to VR thermal design current limit.	
8	Other Status (R0) When set, frequency is reduced below the operating system request due to electrical or other constraints.	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
9	Reserved.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL1.	
11	Package/Platform-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL2/PL3.	
12	Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.	
13	Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
15:14	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Residency State Regulation Log When set, indicates that the Residency State Regulation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR Thermal Design Current Log When set, indicates that the VR TDC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
24	Other Log When set, indicates that the Other Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package or Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package or Platform Level PL2/PL3 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 652H, 1618	MSR_PKG_HDC_CONFIG	
HDC Configuration (R/W)		Package
2:0	PKG_Cx_Monitor Configures Package Cx state threshold for MSR_PKG_HDC_DEEP_RESIDENCY.	
63:3	Reserved.	
Register Address: 653H, 1619	MSR_CORE_HDC_RESIDENCY	
Core HDC Idle Residency (R/O)		Core
63:0	Core_Cx_Duty_Cycle_Cnt	
Register Address: 655H, 1621	MSR_PKG_HDC_SHALLOW_RESIDENCY	
Accumulate the cycles the package was in C2 state and at least one logical processor was in forced idle (R/O)		Package
63:0	Pkg_C2_Duty_Cycle_Cnt	
Register Address: 656H, 1622	MSR_PKG_HDC_DEEP_RESIDENCY	
Package Cx HDC Idle Residency (R/O)		Package
63:0	Pkg_Cx_Duty_Cycle_Cnt	
Register Address: 658H, 1624	MSR_WEIGHTED_CORE_CO	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Core-count Weighted C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.	
Register Address: 659H, 1625	MSR_ANY_CORE_C0	
Any Core C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is one if any processor core in the package is in C0.	
Register Address: 65AH, 1626	MSR_ANY_GFXE_C0	
Any Graphics Engine C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is one if any processor graphic device's compute engines are in C0.	
Register Address: 65BH, 1627	MSR_CORE_GFXE_OVERLAP_C0	
Core and Graphics Engine Overlapped C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is one if at least one compute engine of the processor graphics is in C0 and at least one processor core in the package is also in C0.	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
Platform Power Limit Control (R/W-L) Allows platform BIOS to limit power consumption of the platform devices to the specified values. The Long Duration power consumption is specified via Platform_Power_Limit_1 and Platform_Power_Limit_1_Time. The Short Duration power consumption limit is specified via the Platform_Power_Limit_2 with duration chosen by the processor. The processor implements an exponential-weighted algorithm in the placement of the time windows. Note that the bit field widths differ depending on the platform.		Platform
MSR_PLATFORM_POWER_LIMIT for 4th and 5th generation Intel® Xeon® Scalable Processor Families, Intel® Xeon® 6 E-Core Processors, and Intel® Xeon 6 P-core Processors		
16:0	Platform Power Limit #1 Average Power limit value which the platform must not exceed over a time window as specified by Power_Limit_1_TIME field. The default value is the Thermal Design Power (TDP) and varies with product skus. The unit is specified in MSR_RAPLPOWER_UNIT.	
17	Enable Platform Power Limit #1 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #1 over the time window specified by Power Limit #1 Time Window.	
18	Platform Clamping Limitation #1 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #1 value. This bit is writeable only when CPUID.06H:EAX[4] is set.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
25:19	Time Window for Platform Power Limit #1 Specifies the duration of the time window over which Platform Power Limit 1 value should be maintained for sustained long duration. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[25:24] Y = POWER_LIMIT_1_TIME[23:19] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, and the unit is specified in MSR_RAPL_POWER_UNIT[Time Unit].	
31:26	Reserved.	
48:32	Platform Power Limit #2 Average Power limit value which the platform must not exceed over the Short Duration time window chosen by the processor. The recommended default value is 1.25 times the Long Duration Power Limit (i.e., Platform Power Limit # 1).	
49	Enable Platform Power Limit #2 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #2 over the Short Duration time window.	
50	Platform Clamping Limitation #2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #2 value.	
62:51	Reserved.	
63	Lock. Setting this bit will lock all other bits of this MSR until system RESET.	
MSR_PLATFORM_POWER_LIMIT for 6th–13th Generation Intel® Core™ Processors, 1st–3rd Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, and Intel® Series 2 Core™ Ultra Processors		
14:0	Platform Power Limit #1 Average Power limit value which the platform must not exceed over a time window as specified by Power_Limit_1_TIME field. The default value is the Thermal Design Power (TDP) and varies with product skus. The unit is specified in MSR_RAPLPOWER_UNIT.	
15	Enable Platform Power Limit #1 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #1 over the time window specified by Power Limit #1 Time Window.	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
16	Platform Clamping Limitation #1 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #1 value. This bit is writeable only when CPUID.06H:EAX[4] is set.	
23:17	Time Window for Platform Power Limit #1 Specifies the duration of the time window over which Platform Power Limit 1 value should be maintained for sustained long duration. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, and the unit is specified in MSR_RAPL_POWER_UNIT[Time Unit].	
31:24	Reserved.	
46:32	Platform Power Limit #2 Average Power limit value which the platform must not exceed over the Short Duration time window chosen by the processor. The recommended default value is 1.25 times the Long Duration Power Limit (i.e., Platform Power Limit # 1).	
47	Enable Platform Power Limit #2 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #2 over the Short Duration time window.	
48	Platform Clamping Limitation #2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #2 value.	
62:49	Reserved.	
63	Lock. Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 690H, 1680	MSR_LASTBRANCH_16_FROM_IP	
Last Branch Record 16 From IP (R/W) One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.12.		Thread
Register Address: 691H, 1681	MSR_LASTBRANCH_17_FROM_IP	
Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 692H, 1682	MSR_LASTBRANCH_18_FROM_IP	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 693H, 1683	MSR_LASTBRANCH_19_FROM_IP	
Last Branch Record 19 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 694H, 1684	MSR_LASTBRANCH_20_FROM_IP	
Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 695H, 1685	MSR_LASTBRANCH_21_FROM_IP	
Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 696H, 1686	MSR_LASTBRANCH_22_FROM_IP	
Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 697H, 1687	MSR_LASTBRANCH_23_FROM_IP	
Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 698H, 1688	MSR_LASTBRANCH_24_FROM_IP	
Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 699H, 1689	MSR_LASTBRANCH_25_FROM_IP	
Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69AH, 1690	MSR_LASTBRANCH_26_FROM_IP	
Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69BH, 1691	MSR_LASTBRANCH_27_FROM_IP	
Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69CH, 1692	MSR_LASTBRANCH_28_FROM_IP	
Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69DH, 1693	MSR_LASTBRANCH_29_FROM_IP	
Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69EH, 1694	MSR_LASTBRANCH_30_FROM_IP	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69FH, 1695	MSR_LASTBRANCH_31_FROM_IP	
Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6B0H, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	Running Average Thermal Limit Status (R0) When set, frequency is reduced due to running average thermal limit.	
6	VR Therm Alert Status (R0) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR Thermal Design Current Status (R0) When set, frequency is reduced due to VR TDC limit.	
8	Other Status (R0) When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	Package/Platform-Level PL2 Power Limiting Status (R0) When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
12	Inefficient Operation Status (R0) When set, processor graphics frequency is operating below target frequency.	
15:13	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	
21	Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR Thermal Design Current Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	Other Log When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Inefficient Operation Log When set, indicates that the Inefficient Operation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:29	Reserved.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced due to assertion of external PROCHOT.	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	Thermal Status (R0) When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	Running Average Thermal Limit Status (R0) When set, frequency is reduced due to running average thermal limit.	
6	VR Therm Alert Status (R0) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR Thermal Design Current Status (R0) When set, frequency is reduced due to VR TDC limit.	
8	Other Status (R0) When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) When set, frequency is reduced due to package/Platform-level power limiting PL1.	
11	Package/Platform-Level PL2 Power Limiting Status (R0) When set, frequency is reduced due to package/Platform-level power limiting PL2/PL3.	
15:12	Reserved	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	
21	Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
23	VR Thermal Design Current Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	Other Log When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:28	Reserved.	
Register Address: 6D0H, 1744	MSR_LASTBRANCH_16_TO_IP	
Last Branch Record 16 To IP (R/W) One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.12.		Thread
Register Address: 6D1H, 1745	MSR_LASTBRANCH_17_TO_IP	
Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D2H, 1746	MSR_LASTBRANCH_18_TO_IP	
Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D3H, 1747	MSR_LASTBRANCH_19_TO_IP	
Last Branch Record 19 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D4H, 1748	MSR_LASTBRANCH_20_TO_IP	
Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D5H, 1749	MSR_LASTBRANCH_21_TO_IP	
Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 6D6H, 1750	MSR_LASTBRANCH_22_TO_IP	
Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D7H, 1751	MSR_LASTBRANCH_23_TO_IP	
Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D8H, 1752	MSR_LASTBRANCH_24_TO_IP	
Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D9H, 1753	MSR_LASTBRANCH_25_TO_IP	
Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DAH, 1754	MSR_LASTBRANCH_26_TO_IP	
Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DBH, 1755	MSR_LASTBRANCH_27_TO_IP	
Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DCH, 1756	MSR_LASTBRANCH_28_TO_IP	
Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DDH, 1757	MSR_LASTBRANCH_29_TO_IP	
Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DEH, 1758	MSR_LASTBRANCH_30_TO_IP	
Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DFH, 1759	MSR_LASTBRANCH_31_TO_IP	
Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 770H, 1904	IA32_PM_ENABLE	
See Section 17.4.2, “Enabling HWP.”		Package
Register Address: 771H, 1905	IA32_HWP_CAPABILITIES	
See Section 17.4.3, “HWP Performance Range and Dynamic Capabilities.”		Thread
Register Address: 772H, 1906	IA32_HWP_REQUEST_PKG	
See Section 17.4.4, “Managing HWP.”		Package

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 773H, 1907	IA32_HWP_INTERRUPT	
See Section 17.4.6, “HWP Notifications.”		Thread
Register Address: 774H, 1908	IA32_HWP_REQUEST	
See Section 17.4.4, “Managing HWP.”		Thread
7:0	Minimum Performance (R/W)	
15:8	Maximum Performance (R/W)	
23:16	Desired Performance (R/W)	
31:24	Energy/Performance Preference (R/W)	
41:32	Activity Window (R/W)	
42	Package Control (R/W)	
63:43	Reserved.	
Register Address: 777H, 1911	IA32_HWP_STATUS	
See Section 17.4.5, “HWP Feedback.”		Thread
Register Address: D90H, 3472	IA32_BNDCFGS	
See Table 2-2.		Thread
Register Address: DAOH, 3488	IA32_XSS	
See Table 2-2.		Thread
Register Address: DBOH, 3504	IA32_PKG_HDC_CTL	
See Section 17.5.2, “Package level Enabling HDC.”		Package
Register Address: DB1H, 3505	IA32_PM_CTL1	
See Section 17.5.3, “Logical-Processor Level HDC Control.”		Thread
Register Address: DB2H, 3506	IA32_THREAD_STALL	
See Section 17.5.4.1, “IA32_THREAD_STALL.”		Thread
Register Address: DCOH, 3520	MSR_LBR_INFO_0	
Last Branch Record 0 Additional Information (R/W) One of 32 triplet of last branch record registers on the last branch record stack. This part of the stack contains flag, TSX-related and elapsed cycle information. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 20.9.1, “LBR Stack.”		Thread
Register Address: DC1H, 3521	MSR_LBR_INFO_1	
Last Branch Record 1 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC2H, 3522	MSR_LBR_INFO_2	
Last Branch Record 2 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC3H, 3523	MSR_LBR_INFO_3	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 3 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC4H, 3524	MSR_LBR_INFO_4	
Last Branch Record 4 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC5H, 3525	MSR_LBR_INFO_5	
Last Branch Record 5 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC6H, 3526	MSR_LBR_INFO_6	
Last Branch Record 6 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC7H, 3527	MSR_LBR_INFO_7	
Last Branch Record 7 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC8H, 3528	MSR_LBR_INFO_8	
Last Branch Record 8 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC9H, 3529	MSR_LBR_INFO_9	
Last Branch Record 9 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCAH, 3530	MSR_LBR_INFO_10	
Last Branch Record 10 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCBH, 3531	MSR_LBR_INFO_11	
Last Branch Record 11 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCCH, 3532	MSR_LBR_INFO_12	
Last Branch Record 12 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCDH, 3533	MSR_LBR_INFO_13	
Last Branch Record 13 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCEH, 3534	MSR_LBR_INFO_14	
Last Branch Record 14 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCFH, 3535	MSR_LBR_INFO_15	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 15 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDOH, 3536	MSR_LBR_INFO_16	
Last Branch Record 16 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD1H, 3537	MSR_LBR_INFO_17	
Last Branch Record 17 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD2H, 3538	MSR_LBR_INFO_18	
Last Branch Record 18 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD3H, 3539	MSR_LBR_INFO_19	
Last Branch Record 19 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD4H, 3540	MSR_LBR_INFO_20	
Last Branch Record 20 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD5H, 3541	MSR_LBR_INFO_21	
Last Branch Record 21 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD6H, 3542	MSR_LBR_INFO_22	
Last Branch Record 22 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD7H, 3543	MSR_LBR_INFO_23	
Last Branch Record 23 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD8H, 3544	MSR_LBR_INFO_24	
Last Branch Record 24 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD9H, 3545	MSR_LBR_INFO_25	
Last Branch Record 25 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDAH, 3546	MSR_LBR_INFO_26	
Last Branch Record 26 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDBH, 3547	MSR_LBR_INFO_27	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 27 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDCH, 3548	MSR_LBR_INFO_28	
Last Branch Record 28 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDDH, 3549	MSR_LBR_INFO_29	
Last Branch Record 29 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDEH, 3550	MSR_LBR_INFO_30	
Last Branch Record 30 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDFH, 3551	MSR_LBR_INFO_31	
Last Branch Record 31 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread

Table 2-40 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_4EH, 06_5EH, 06_8EH, 06_9EH, or 06_66H.

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
43:0	Current count.	
63:44	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).	
63:4	Reserved.	

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTRO	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSELO	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb Unit, Counter 1 Event Select MSR		Package
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTRO	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTRO	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTRO	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTRO	

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: E01H, 3585	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	
3	Slice 3 select.	
4	Slice 4select.	
18:5	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: E02H, 3586	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

2.17.1 MSRs Introduced in 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Table 2-41 lists additional MSRs for 7th generation and 8th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_8EH or 06_9EH. For an MSR listed in Table 2-41 that also appears in the model-specific tables of prior generations, Table 2-41 supersedes prior generation tables.

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 80H, 128	MSR_TRACE_HUB_STH ACPIBAR_BASE	
NPK Address Used by AET Messages (R/W)		Package
0	Lock Bit If set, this MSR cannot be re-written anymore. Lock bit has to be set in order for the AET packets to be directed to NPK MMIO.	

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
17:1	Reserved.	
63:18	ACPIBAR_BASE_ADDRESS AET target address in NPK MMIO space.	
Register Address: 1F4H, 500	MSR_PRMRR_PHYS_BASE	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MemType PRMRR BASE MemType.	
11:3	Reserved.	
45:12	Base PRMRR Base Address.	
63:46	Reserved.	
Register Address: 1F5H, 501	MSR_PRMRR_PHYS_MASK	
Processor Reserved Memory Range Register - Physical Mask Control Register (R/W)		Core
9:0	Reserved.	
10	Lock Lock bit for the PRMRR.	
11	VLD Enable bit for the PRMRR.	
45:12	Mask PRMRR MASK bits.	
63:46	Reserved.	
Register Address: 1FBH, 507	MSR_PRMRR_VALID_CONFIG	
Valid PRMRR Configurations (R/W)		Core
0	1M supported MEE size.	
4:1	Reserved.	
5	32M supported MEE size.	
6	64M supported MEE size.	
7	128M supported MEE size.	
31:8	Reserved.	
Register Address: 2F4H, 756	MSR_UNCORE_PRMRR_PHYS_BASE ¹	
(R/W)	The PRMRR range is used to protect the processor reserved memory from unauthorized reads and writes. Any IO access to this range is aborted. This register controls the location of the PRMRR range by indicating its starting address. It functions in tandem with the PRMRR mask register.	
11:0	Reserved.	Package
PAWIDTH-1:12	Range Base This field corresponds to bits PAWIDTH-1:12 of the base address memory range which is allocated to PRMRR memory.	
63:PAWIDTH	Reserved.	

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2F5H, 757	MSR_UNCORE_PRMR_PHYS_MASK ¹	
(R/W) This register controls the size of the PRMRR range by indicating which address bits must match the PRMRR base register value.		Package
9:0	Reserved.	
10	Lock Setting this bit locks all writeable settings in this register, including itself.	
11	Range_En Indicates whether the PRMRR range is enabled and valid.	
38:12	Range_Mask This field indicates which address bits must match PRMRR base in order to qualify as an PRMRR access.	
63:39	Reserved.	
Register Address: 620H, 1568	MSR_RING_RATIO_LIMIT	
Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.		Package
6:0	MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	

NOTES:

1. This MSR is specific to 7th generation and 8th generation Intel® Core™ processors.

2.17.2 MSRs Specific to 8th Generation Intel® Core™ i3 Processors

Table 2-42 lists additional MSRs for 8th generation Intel Core i3 processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_66H. For an MSR listed in Table 2-42 that also appears in the model-specific tables of prior generations, Table 2-42 supersedes prior generation tables.

Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	

**Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	Enable VMX Outside SMX Operation (R/WL)	
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
17	SGX Launch Control Enable (R/WL) This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR. Available only if CPUID.07H.00H:ECX[30] = 1.	
18	SGX Global Functions Enable (R/WL)	
63:21	Reserved.	
Register Address: 350H, 848	MSR_BR_DETECT_CTRL	
Branch Monitoring Global Control (R/W)		
0	EnMonitoring Global enable for branch monitoring.	
1	EnExcept Enable branch monitoring event signaling on threshold trip. The branch monitoring event handler is signaled via the existing PMI signaling mechanism as programmed from the corresponding local APIC LVT entry.	
2	EnLBRFrz Enable LBR freeze on threshold trip. This will cause the LBR frozen bit 58 to be set in IA32_PERF_GLOBAL_STATUS when a triggering condition occurs and this bit is enabled.	
3	DisableInGuest When set to '1', branch monitoring, event triggering and LBR freeze actions are disabled when operating at VMX non-root operation.	
7:4	Reserved.	
17:8	WindowSize Window size defined by WindowCntSel. Values 0 - 1023 are supported. Once the Window counter reaches the WindowSize count both the Window Counter and all Branch Monitoring Counters are cleared.	
23:18	Reserved.	
25:24	WindowCntSel Window event count select: '00 = Instructions retired. '01 = Branch instructions retired '10 = Return instructions retired. '11 = Indirect branch instructions retired.	

**Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
26	CntAndMode When set to '1', the overall branch monitoring event triggering condition is true only if all enabled counters' threshold conditions are true. When '0', the threshold tripping condition is true if any enabled counters' threshold is true.	
63:27	Reserved.	
Register Address: 351H, 849	MSR_BR_DETECT_STATUS	
Branch Monitoring Global Status (R/W)		
0	Branch Monitoring Event Signaled When set to '1', Branch Monitoring event signaling is blocked until this bit is cleared by software.	
1	LBRsValid This status bit is set to '1' if the LBR state is considered valid for sampling by branch monitoring software.	
7:2	Reserved.	
8	CntrHit0 Branch monitoring counter #0 threshold hit. This status bit is sticky and once set requires clearing by software. Counter operation continues independent of the state of the bit.	
9	CntrHit1 Branch monitoring counter #1 threshold hit. This status bit is sticky and once set requires clearing by software. Counter operation continues independent of the state of the bit.	
15:10	Reserved. Reserved for additional branch monitoring counters threshold hit status.	
25:16	CountWindow The current value of the window counter. The count value is frozen on a valid branch monitoring triggering condition. This is a 10-bit unsigned value.	
31:26	Reserved. Reserved for future extension of CountWindow.	
39:32	Count0 The current value of counter 0 updated after each occurrence of the event being counted. The count value is frozen on a valid branch monitoring triggering condition (in which case CntrHit0 will also be set). This is an 8-bit signed value (2's complement). Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256). RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).	

**Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
47:40	Count1 The current value of counter 1 updated after each occurrence of the event being counted. The count value is frozen on a valid branch monitoring triggering condition (in which case CntrHit1 will also be set). This is an 8-bit signed value (2's complement). Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256). RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).	
63:48	Reserved.	
Register Address: 354H–355H, 852–853	MSR_BR_DETECT_COUNTER_CONFIG_i	
Branch Monitoring Detect Counter Configuration (R/W)		
0	CntrEn Enable counter.	
7:1	CntrEvSel Event select (other values #GP) '0000000 = RETs. '0000001 = RET-CALL bias. '0000010 = RET mispredicts. '0000011 = Branch (all) mispredicts. '0000100 = Indirect branch mispredicts. '0000101 = Far branch instructions.	
14:8	CntrThreshold Threshold (an unsigned value of 0 to 127 supported). The value 0 of counter threshold will result in event signaled after every instruction. #GP if threshold is < 2.	
15	MispredEventCnt Mispredict events counting behavior: '0 = Mispredict events are counted in a window. '1 = Mispredict events are counted based on a consecutive occurrence. CntrThreshold is treated as # of consecutive mispredicts. This control bit only applies to events specified by CntrEvSel that involve a prediction (0000010, 0000011, 0000100). Setting this bit for other events is ignored.	
63:16	Reserved.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Package C3 Residency Counter (R/O)		Package
63:0	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.	
Register Address: 620H, 1568	MSR_RING_RATIO_LIMIT	
Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.		Package

Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
6:0	MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	
Register Address: 660H, 1632	MSR_CORE_C1_RESIDENCY	
Core C1 Residency Counter (R/O)		Core
63:0	Value since last reset for the Core C1 residency. Counter rate is the Max Non-Turbo frequency (same as TSC). This counter counts in case both of the core's threads are in an idle state and at least one of the core's thread residency is in a C1 state or in one of its sub states. The counter is updated only after a core C state exit. Note: Always reads 0 if core C1 is unsupported. A value of zero indicates that this processor does not support core C1 or never entered core C1 level state.	
Register Address: 662H, 1634	MSR_CORE_C3_RESIDENCY	
Core C3 Residency Counter (R/O)		Core
63:0	Will always return 0.	

Table 2-43 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_66H.

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Report the number of C-Box units with performance counters, including processor cores and processor graphics.	

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSELO	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb unit, Counter 1 Event Select MSR		Package
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 702H, 1794	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 703H, 1795	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 708H, 1800	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 709H, 1801	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 70AH, 1802	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 70BH, 1803	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 712H, 1810	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 713H, 1811	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 718H, 1816	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 719H, 1817	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 71AH, 1818	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 71BH, 1819	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_4_PERFEVTSELO	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package
Register Address: 722H, 1826	MSR_UNC_CBO_4_PERFCTR0	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 723H, 1827	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 728H, 1832	MSR_UNC_CBO_5_PERFEVTSELO	
Uncore C-Box 5, Counter 0 Event Select MSR		Package
Register Address: 729H, 1833	MSR_UNC_CBO_5_PERFEVTSEL1	
Uncore C-Box 5, Counter 1 Event Select MSR		Package
Register Address: 72AH, 1834	MSR_UNC_CBO_5_PERFCTR0	
Uncore C-Box 5, Performance Counter 0		Package
Register Address: 72BH, 1835	MSR_UNC_CBO_5_PERFCTR1	
Uncore C-Box 5, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_6_PERFEVTSELO	
Uncore C-Box 6, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_6_PERFEVTSEL1	
Uncore C-Box 6, Counter 1 Event Select MSR		Package
Register Address: 732H, 1842	MSR_UNC_CBO_6_PERFCTR0	
Uncore C-Box 6, Performance Counter 0		Package
Register Address: 733H, 1843	MSR_UNC_CBO_6_PERFCTR1	
Uncore C-Box 6, Performance Counter 1		Package
Register Address: 738H, 1848	MSR_UNC_CBO_7_PERFEVTSELO	
Uncore C-Box 7, Counter 0 Event Select MSR		Package
Register Address: 739H, 1849	MSR_UNC_CBO_7_PERFEVTSEL1	
Uncore C-Box 7, Counter 1 Event Select MSR		Package
Register Address: 73AH, 1850	MSR_UNC_CBO_7_PERFCTR0	
Uncore C-Box 7, Performance Counter 0		Package
Register Address: 73BH, 1851	MSR_UNC_CBO_7_PERFCTR1	
Uncore C-Box 7, Performance Counter 1		Package
Register Address: E01H, 3585	MSR_UNC_PERF_GLOBAL_CTRL	

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	
3	Slice 3 select.	
4	Slice 4select.	
18:5	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: E02H, 3586	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

2.17.3 MSRs Introduced in 10th Generation Intel® Core™ Processors

Table 2-44 lists additional MSRs for 10th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_7EH. For an MSR listed in Table 2-44 that also appears in the model-specific tables of prior generations, Table 2-44 supersedes prior generation tables.

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
28:0	Reserved.	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 11.1.2.3, "Features to Disable Bus Locks."	
30	Reserved.	
31	Reserved.	
Register Address: 48H, 72	IA32_SPEC_CTRL	
See Table 2-2.		Core
Register Address: 49H, 73	IA32_PREDICT_CMD	

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Thread
Register Address: 8CH, 140	IA32_SGXLEPUBKEYHASH0	
See Table 2-2.		Thread
Register Address: 8DH, 141	IA32_SGXLEPUBKEYHASH1	
See Table 2-2.		Thread
Register Address: 8EH, 142	IA32_SGXLEPUBKEYHASH2	
See Table 2-2.		Thread
Register Address: 8FH, 143	IA32_SGXLEPUBKEYHASH3	
See Table 2-2.		Thread
Register Address: A0H, 160	MSR_BIOS_MCU_ERRORCODE	
BIOS MCU ERRORCODE (R/O) This MSR indicates if WRMSR 0x79 failed to configure PRM memory and gives a hint to debug BIOS.		Package
15:0	Error Codes (R/O)	Package
30:16	Reserved.	
31	MCU Partial Success (R/O) When set to 1, WRMSR 0x79 skipped part of the functionality during BIOS.	Thread
Register Address: A5H, 165	MSR_FIT_BIOS_ERROR	
FIT BIOS ERROR (R/W) Report error codes for debug in case the processor failed to parse the Firmware Table in BIOS. Can also be used to log BIOS information.		Thread
7:0	Error Codes (R/W) Error codes for debug.	
15:8	Entry Type (R/W) Failed FIT entry type.	
16	FIT MCU Entry (R/W) FIT contains MCU entry.	
62:17	Reserved.	
63	LOCK (R/W) When set to 1, writes to this MSR will be skipped.	
Register Address: 10BH, 267	IA32_FLUSH_CMD	
See Table 2-2.		Thread
Register Address: 151H, 337	MSR_BIOS_DONE	
BIOS Done (R/WO)		Thread
0	BIOS Done Indication (R/WO) Set by BIOS when it finishes programming the processor and wants to lock the memory configuration from changes by software that is running on this thread. Writes to the bit will be ignored if EAX[0] is 0.	Thread

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	Package BIOS Done Indication (R/O) When set to 1, all threads in the package have bit 0 of this MSR set.	Package
31:2	Reserved.	
Register Address: 1F1H, 497	MSR_CRASHLOG_CONTROL	
Write Data to a Crash Log Configuration		Thread
0	CDDIS: CrashDump_Disable If set, indicates that Crash Dump is disabled.	
63:1	Reserved.	
Register Address: 2A0H, 672	MSR_PRMRR_BASE_0	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE: PRMRR BASE Memory Type.	
3	CONFIGURED: PRMRR BASE Configured.	
11:4	Reserved.	
51:12	BASE: PRMRR Base Address.	
63:52	Reserved.	
Register Address: 30CH, 780	IA32_FIXED_CTR3	
Fixed-Function Performance Counter Register 3 (R/W) Bit definitions are the same as found in IA32_FIXED_CTR0, offset 309H. See Table 2-2.		Thread
Register Address: 329H, 809	MSR_PERF_METRICS	
Performance Metrics (R/W) Reports metrics directly. Software can check (and/or expose to its guests) the availability of PERF_METRICS feature using IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE (bit 15).		Thread
7:0	Retiring. Percent of utilized slots by uops that eventually retire (commit).	
15:8	Bad Speculation. Percent of wasted slots due to incorrect speculation, covering utilized by uops that do not retire, or recovery bubbles (unutilized slots).	
23:16	Frontend Bound. Percent of unutilized slots where front-end did not deliver a uop while back-end is ready.	
31:24	Backend Bound. Percent of unutilized slots where a uop was not delivered to back-end due to lack of back-end resources.	
63:32	Reserved.	
Register Address: 3F2H, 1010	MSR_PEBS_DATA_CFG	
PEBS Data Configuration (R/W) Provides software the capability to select data groups of interest and thus reduce the record size in memory and record generation latency. Hence, a PEBS record's size and layout vary based on the selected groups. The MSR also allows software to select LBR depth for branch data records.		Thread
0	Memory Info. Setting this bit will capture memory information such as the linear address, data source and latency of the memory access in the PEBS record.	

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	GPRs. Setting this bit will capture the contents of the General Purpose registers in the PEBS record.	
2	XMMs. Setting this bit will capture the contents of the XMM registers in the PEBS record.	
3	LBRs. Setting this bit will capture LBR TO, FROM, and INFO in the PEBS record.	
23:4	Reserved.	
31:24	LBR Entries. Set the field to the desired number of entries - 1. For example, if the LBR_entries field is 0, a single entry will be included in the record. To include 32 LBR entries, set the LBR_entries field to 31 (0x1F). To ensure all PEBS records are 16-byte aligned, software can use LBR_entries that is multiple of 3.	
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W)		Core
0	L1 Scrubbing Enable When set to 1, enable L1 scrubbing.	
31:1	Reserved.	
Register Address: 657H, 1623	MSR_FAST_UNCORE_MSRS_CTL	
Fast WRMSR/RDMSR Control MSR (R/W)		Thread
3:0	FAST_ACCESS_ENABLE: Bit 0: When set to '1', provides a hint for the hardware to enable fast access mode for the IA32_HWP_REQUEST MSR. This bit is sticky and is cleaned by the hardware only during reset time. This bit is valid only if FAST_UNCORE_MSRS_CAPABILITY[0] is set. Setting this bit will cause CPUID.06H:EAX[18] to be set.	
31:4	Reserved.	
Register Address: 65EH, 1630	MSR_FAST_UNCORE_MSRS_STATUS	
Indication of Uncore MSRs, Post Write Activates		Thread
0	Indicates whether the CPU is still in the middle of writing IA32_HWP_REQUEST MSR, even after the WRMSR instruction has retired. A value of 1 indicates the last write of IA32_HWP_REQUEST is still ongoing. A value of 0 indicates the last write of IA32_HWP_REQUEST is visible outside the logical processor. Software can use the status of this bit to avoid overwriting IA32_HWP_REQUEST.	
31:1	Reserved.	
Register Address: 65FH, 1631	MSR_FAST_UNCORE_MSRS_CAPABILITY	
Fast WRMSR/RDMSR Enumeration MSR (R/O)		Thread

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3:0	MSRS_CAPABILITY: Bit 0: If set to '1', hardware supports the fast access mode for the IA32_HWP_REQUEST MSR.	
31:4	Reserved.	
Register Address: 772H, 1906	IA32_HWP_REQUEST_PKG	
See Table 2-2.		Package
Register Address: 775H, 1909	IA32_PECI_HWP_REQUEST_INFO	
See Table 2-2.		Package
Register Address: 777H, 1911	IA32_HWP_STATUS	
See Table 2-2.		Thread

2.17.4 MSRs Introduced in the 11th Generation Intel® Core™ Processors based on Tiger Lake Microarchitecture

Table 2-45 lists additional MSRs for 11th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_8CH or 06_8DH. The MSRs listed in Table 2-44 are also supported by these processors. For an MSR listed in Table 2-45 that also appears in the model-specific tables of prior generations, Table 2-45 supersedes prior generation tables.

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: A0H, 160	MSR_BIOS_MCU_ERRORCODE	
BIOS MCU ERRORCODE (R/O)		Package
15:0	Error Codes	
31:16	Reserved.	
Register Address: A7H, 167	MSR_BIOS_DEBUG	
BIOS DEBUG (R/O) This MSR indicates if WRMSR 79H failed to configure PRM memory and gives a hint to debug BIOS.		Thread
30:0	Reserved.	
31	MCU Partial Success When set to 1, WRMSR 79H skipped part of the functionality during BIOS.	
63:32	Reserved.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/O) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Package
1:0	Reserved.	
2	FUSA_SUPPORTED	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3	RSM_IN_CPL0_ONLY When set to 1, the RSM instruction is only allowed in CPL0 (#GP triggered in any CPL != 0). When set to 0, then any CPL may execute the RSM instruction.	
4	Reserved.	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL (MSR address 33H).	
31:6	Reserved.	
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTL3	
IA32_VMX_PROCBASED_CTL3 This MSR enumerates the allowed 1-settings of the third set of processor-based controls. Specifically, VM entry allows bit X of the tertiary processor-based VM-execution controls to be 1 if and only if bit X of the MSR is set to 1. If bit X of the MSR is cleared to 0, VM entry fails if control X and the “activate tertiary controls” primary processor-based VM-execution control are both 1.		Core
0	LOADIWKEY This control determines whether executions of LOADIWKEY cause VM exits.	
63:1	Reserved.	
Register Address: 601H, 1537	MSR_VR_CURRENT_CONFIG	
Power Limit 4 (PL4) Package-level maximum power limit (in Watts). It is a proactive, instantaneous limit.		Package
12:0	PL4 Value PL4 value in 0.125 A increments. This field is locked by VR_CURRENT_CONFIG[LOCK]. When the LOCK bit is set to 1b, this field becomes Read Only.	
30:13	Reserved.	
31	Lock Indication (LOCK) This bit will lock the CURRENT_LIMIT settings in this register and will also lock this setting. This means that once set to 1b, the CURRENT_LIMIT setting and this bit become Read Only until the next Warm Reset.	
62:32	Not in use.	
63	Reserved.	
Register Address: 6A0H, 1696	IA32_U_CET	
Configure User Mode CET (R/W) See Table 2-2.		
Register Address: 6A2H, 1698	IA32_S_CET	
Configure Supervisor Mode CET (R/W) See Table 2-2.		
Register Address: 6A4H, 1700	IA32_PLO_SSP	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Linear address to be loaded into SSP on transition to privilege level 0. (R/W) See Table 2-2.		
Register Address: 6A5H, 1701	IA32_PL1_SSP	
Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.		
Register Address: 6A6H, 1702	IA32_PL2_SSP	
Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.		
Register Address: 6A7H, 1703	IA32_PL3_SSP	
Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.		
Register Address: 6A8H, 1704	IA32_INTERRUPT_SSP_TABLE_ADDR	
Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) See Table 2-2.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
See Table 2-2.		
Register Address: 982H, 2434	IA32_TME_ACTIVATE	
See Table 2-2.		
Register Address: 983H, 2435	IA32_TME_EXCLUDE_MASK	
See Table 2-2.		
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
See Table 2-2.		
Register Address: 990H, 2448	IA32_COPY_STATUS ¹	
See Table 2-2.		Thread
Register Address: 991H, 2449	IA32_IWKEYBACKUP_STATUS ¹	
See Table 2-2.		Platform
Register Address: C82H, 3202	IA32_L2_QOS_CFG	
IA32_CR_L2_QOS_CFG This MSR provides software an enumeration of the parameters that L2 QoS (Intel RDT) support in any particular implementation.		Core
0	CDP_ENABLE When set to 1, it will enable the code and data prioritization for the L2 CAT/Intel RDT feature. When set to 0, code and data prioritization is disabled for L2 CAT/Intel RDT. See Chapter 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features,” for further details on CDP.	
31:1	Reserved.	
Register Address: D10H–D17H, 3220–3351	IA32_L2_QOS_MASK [0-7]	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
IA32_CR_L2_QOS_MASK[0-7] Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Package
19:0	WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.04H.02H:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.	
31:20	Reserved.	
Register Address: D91H, 3473	IA32_COPY_LOCAL_TO_PLATFORM ¹	
See Table 2-2.		Thread
Register Address: D92H, 3474	IA32_COPY_PLATFORM_TO_LOCAL ¹	
See Table 2-2.		Thread

NOTES:

1. Further details on Key Locker and usage of this MSR can be found here:

<https://software.intel.com/content/www/us/en/develop/download/intel-key-locker-specification.html>.

2.17.5 MSRs Introduced in the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Table 2-46 lists additional MSRs for 12th and 13th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_97H, 06_9AH, 06_BAH, 06_B7H, or 06_BFH. Table 2-47 lists the MSRs unique to the processor P-core. Table 2-48 lists the MSRs unique to the processor E-core.

The MSRs listed in Table 2-44¹ and Table 2-45 are also supported by these processors. For an MSR listed in Table 2-46, Table 2-47, or Table 2-48 that also appears in the model-specific tables of prior generations, Table 2-46, Table 2-47, and Table 2-48 supersede prior generation tables.

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor will only allow one in-progress UC store at a time.	

1. MSRs at the following addresses are not supported in the 12th and 13th generation Intel Core processor E-core: 30CH, 329H, 541H, and 657H. The MSR at address 657H is not supported in the 12th and 13th generation Intel Core processor P-core.

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
28	UC_LOCK_DISABLE If set to 1, a UC lock will cause a #GP(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
30	Reserved.	
31	Reserved.	
Register Address: BCH, 188	IA32_MISC_PACKAGE_CTL5	
Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.		Package
Register Address: C7H, 199	IA32_PMC6	
General Performance Counter 6 (R/W) See Table 2-2.		Core
Register Address: C8H, 200	IA32_PMC7	
General Performance Counter 7 (R/W) See Table 2-2.		Core
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/O) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Package
0	STLB_QOS_SUPPORTED When set to 1, the STLB QoS feature is supported and the STLB QoS MSRs (1A8FH - 1A97H) are accessible. When set to 0, access to these MSRs will #GP.	
1	Reserved.	
2	FUSA_SUPPORTED	
3	RSM_IN_CPLO_ONLY When set to 1, the RSM instruction is only allowed in CPLO (#GP triggered in any CPL != 0). When set to 0, then any CPL may execute the RSM instruction.	
4	UC_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 28 of MSR_MEMORY_CTRL (MSR address 33H).	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL.	
6	SNOOP_FILTER_QOS_SUPPORTED When set to 1, the Snoop Filter Qos Mask MSRs are supported. When set to 0, access to these MSRs will #GP.	

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
7	UC_STORE_THROTTLING_SUPPORTED When set 1, UC Store throttle capability exist through MSR_MEMORY_CTRL (33H) bit 27.	
31:8	Reserved.	
Register Address: E1H, 225	IA32_UMWAIT_CONTROL	
UMWAIT Control (R/W) See Table 2-2.		
Register Address: 10AH, 266	IA32_ARCH_CAPABILITIES	
Enumeration of Architectural Features (R/O) See Table 2-2.		
Register Address: 18CH, 396	IA32_PERFEVTSEL6	
See Table 2-20.		Core
Register Address: 18DH, 397	IA32_PERFEVTSEL7	
See Table 2-20.		Core
Register Address: 195H, 405	IA32_OVERCLOCKING_STATUS	
Overclocking Status (R/O) IA32_ARCH_CAPABILITIES[bit 23] enumerates support for this MSR. See Table 2-2.		Package
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	
Primary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 493H, 1171	IA32_VMX_EXIT_CTL2	
See Table 2-2.		

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4C7H, 1223	IA32_A_PMC6	
Full Width Writable IA32_PMC6 Alias (R/W) See Table 2-2.		
Register Address: 4C8H, 1224	IA32_A_PMC7	
Full Width Writable IA32_PMC7 Alias (R/W) See Table 2-2.		
Register Address: 650H, 1616	MSR_SECONDARY_TURBO_RATIO_LIMIT	
Secondary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Module
63:0	Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.	
Register Address: 6E1H, 1761	IA32_PKRS	
Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.		
Register Address: 776H, 1910	IA32_HWP_CTL	
See Table 2-2.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Memory Encryption Capability MSR See Table 2-2.		
Register Address: 1200H—121FH, 4608—4639	IA32_LBR_x_INFO	
Last Branch Record Entry X Info Register (R/W) See Table 2-2.		
Register Address: 14CEH, 5326	IA32_LBR_CTL	
Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.		
Register Address: 14CFH, 5327	IA32_LBR_DEPTH	
Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.		
Register Address: 1500H—151FH, 5376—5407	IA32_LBR_x_FROM_IP	
Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.		
Register Address: 1600H—161FH, 5632—5663	IA32_LBR_x_TO_IP	
Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.		
Register Address: 17D2H, 6098	IA32_THREAD_FEEDBACK_CHAR	
Thread Feedback Characteristics (R/O) See Table 2-2.		
Register Address: 17D4H, 6100	IA32_HW_FEEDBACK_THREAD_CONFIG	
Hardware Feedback Thread Configuration (R/W) See Table 2-2.		
Register Address: 17DAH, 6106	IA32_HRESET_ENABLE	
History Reset Enable (R/W) See Table 2-2.		

The MSRs listed in Table 2-47 are unique to the 12th and 13th generation Intel Core processor P-core. These MSRs are not supported on the processor E-core.

Table 2-47. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A4H, 420	MSR_PREFETCH_CONTROL	
Prefetch Disable Bits (R/W)		
0	L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	

Table 2-47. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	L2_ADJACENT_CACHE_LINE_PREFETCHER_DISABLE If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	
2	DCU_HARDWARE_PREFETCHER_DISABLE If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	
3	DCU_IP_PREFETCHER_DISABLE If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	
4	Reserved.	
5	AMP_PREFETCH_DISABLE If 1, disables the L2 Adaptive Multipath Probability (AMP) prefetcher.	
63:6	Reserved.	
Register Address: 3F7H, 1015	MSR_PEBS_FRONTEND	
FrontEnd Precise Event Condition Select (R/W) See Table 2-39.		Thread
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W)		Thread
0	WB_MEM_STRM_LD_DISABLE Disable streaming behavior for MOVNTDQA loads to WB memory type. If set, these accesses will be treated like regular cacheable loads (Data will be cached).	
63:1	Reserved.	
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W) See Table 2-44.		Core
Register Address: D10H–D17H, 3220–3351	IA32_L2_QOS_MASK[0-7]	
IA32_CR_L2_QOS_MASK[0-7] If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Core
19:0	WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.04H.02H:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.	
31:20	Reserved.	

The MSRs listed in Table 2-48 are unique to the 12th and 13th generation Intel Core processor E-core. These MSRs are not supported on the processor P-core.

Table 2-48. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor E-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D10H–D1FH, 3220–3359	IA32_L2_QOS_MASK [0-15]	
IA32_CR_L2_QOS_MASK [0-15] If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Module
19:0	WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.04H.02H:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.	
31:20	Reserved.	
Register Address: 1309H–130BH, 4873–4875	MSR_RELOAD_FIXED_CTRx	
Reload value for IA32_FIXED_CTRx (R/W)		
47:0	Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBB_ENABLE, and FIXED_CTRx is overflowed.	
63:48	Reserved.	
Register Address: 14C1H–14C6H, 5313–5318	MSR_RELOAD_PMCx	
Reload value for IA32_PMCx (R/W)		Core
47:0	Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBB_ENABLE, and PMCx is overflowed.	
63:48	Reserved.	

Table 2-49 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_97H, 06_9AH, 06_BAH, 06_B7H, or 06_BFH.

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).	
63:4	Reserved.	
Register Address: 2000H, 8192	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 2001H, 8193	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2002H, 8194	MSR_UNC_CBO_0_PERFCTRO	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 2003H, 8195	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 2008H, 8200	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 2009H, 8201	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 200AH, 8202	MSR_UNC_CBO_1_PERFCTRO	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 200BH, 8203	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 2010H, 8208	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 2011H, 8209	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 2012H, 8210	MSR_UNC_CBO_2_PERFCTRO	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 2013H, 8211	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 2018H, 8216	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 2019H, 8217	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 201AH, 8218	MSR_UNC_CBO_3_PERFCTRO	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 201BH, 8219	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: 2020H, 8224	MSR_UNC_CBO_4_PERFEVTSELO	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 2021H, 8225	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package
Register Address: 2022H, 8226	MSR_UNC_CBO_4_PERFCTRO	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 2023H, 8227	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 2028H, 8232	MSR_UNC_CBO_5_PERFEVTSELO	

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 5, Counter 0 Event Select MSR		Package
Register Address: 2029H, 8233	MSR_UNC_CBO_5_PERFEVTSEL1	
Uncore C-Box 5, Counter 1 Event Select MSR		Package
Register Address: 202AH, 8234	MSR_UNC_CBO_5_PERFCTRO	
Uncore C-Box 5, Performance Counter 0		Package
Register Address: 202BH, 8235	MSR_UNC_CBO_5_PERFCTR1	
Uncore C-Box 5, Performance Counter 1		Package
Register Address: 2030H, 8240	MSR_UNC_CBO_6_PERFEVTSELO	
Uncore C-Box 6, Counter 0 Event Select MSR		Package
Register Address: 2031H, 8241	MSR_UNC_CBO_6_PERFEVTSEL1	
Uncore C-Box 6, Counter 1 Event Select MSR		Package
Register Address: 2032H, 8242	MSR_UNC_CBO_6_PERFCTRO	
Uncore C-Box 6, Performance Counter 0		Package
Register Address: 2033H, 8243	MSR_UNC_CBO_6_PERFCTR1	
Uncore C-Box 6, Performance Counter 1		Package
Register Address: 2038H, 8248	MSR_UNC_CBO_7_PERFEVTSELO	
Uncore C-Box 7, Counter 0 Event Select MSR		Package
Register Address: 2039H, 8249	MSR_UNC_CBO_7_PERFEVTSEL1	
Uncore C-Box 7, Counter 1 Event Select MSR		Package
Register Address: 203AH, 8250	MSR_UNC_CBO_7_PERFCTRO	
Uncore C-Box 7, Performance Counter 0		Package
Register Address: 203BH, 8251	MSR_UNC_CBO_7_PERFCTR1	
Uncore C-Box 7, Performance Counter 1		Package
Register Address: 2040H, 8256	MSR_UNC_CBO_8_PERFEVTSELO	
Uncore C-Box 8, Counter 0 Event Select MSR		Package
Register Address: 2041H, 8257	MSR_UNC_CBO_8_PERFEVTSEL1	
Uncore C-Box 8, Counter 1 Event Select MSR		Package
Register Address: 2042H, 8258	MSR_UNC_CBO_8_PERFCTRO	
Uncore C-Box 8, Performance Counter 0		Package
Register Address: 2043H, 8259	MSR_UNC_CBO_8_PERFCTR1	
Uncore C-Box 8, Performance Counter 1		Package
Register Address: 2048H, 8264	MSR_UNC_CBO_9_PERFEVTSELO	
Uncore C-Box 9, Counter 0 Event Select MSR		Package
Register Address: 2049H, 8265	MSR_UNC_CBO_9_PERFEVTSEL1	
Uncore C-Box 9, Counter 1 Event Select MSR		Package
Register Address: 204AH, 8266	MSR_UNC_CBO_9_PERFCTRO	
Uncore C-Box 9, Performance Counter 0		Package

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 204BH, 8267	MSR_UNC_CBO_9_PERFCTR1	
Uncore C-Box 9, Performance Counter 1		Package
Register Address: 2FD0H, 12240	MSR_UNC_ARB_0_PERFEVTSELO	
Uncore Arb Unit 0, Counter 0 Event Select MSR		Package
Register Address: 2FD1H, 12241	MSR_UNC_ARB_0_PERFEVTSEL1	
Uncore Arb Unit 0, Counter 1 Event Select MSR		Package
Register Address: 2FD2H, 12242	MSR_UNC_ARB_0_PERFCTRO	
Uncore Arb Unit 0, Performance Counter 0		Package
Register Address: 2FD3H, 12243	MSR_UNC_ARB_0_PERFCTR1	
Uncore Arb Unit 0, Performance Counter 1		Package
Register Address: 2FD4H, 12244	MSR_UNC_ARB_0_PERF_STATUS	
Uncore Arb Unit 0, Performance Status		Package
Register Address: 2FD5H, 12245	MSR_UNC_ARB_0_PERF_CTRL	
Uncore Arb Unit 0, Performance Control		Package
Register Address: 2FD8H, 12248	MSR_UNC_ARB_1_PERFEVTSELO	
Uncore Arb Unit 1, Counter 0 Event Select MSR		Package
Register Address: 2FD9H, 12249	MSR_UNC_ARB_1_PERFEVTSEL1	
Uncore Arb Unit 1, Counter 1 Event Select MSR		Package
Register Address: 2FDAH, 12250	MSR_UNC_ARB_1_PERFCTRO	
Uncore Arb Unit 1, Performance Counter 0		Package
Register Address: 2FDBH, 12251	MSR_UNC_ARB_1_PERFCTR1	
Uncore Arb Unit 1, Performance Counter 1		Package
Register Address: 2FDCH, 12252	MSR_UNC_ARB_1_PERF_STATUS	
Uncore Arb Unit 1, Performance Status		Package
Register Address: 2FDDH, 12253	MSR_UNC_ARB_1_PERF_CTRL	
Uncore Arb Unit 1, Performance Control		Package
Register Address: 2FDEH, 12254	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 2FDFH, 12255	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
43:0	Current count.	
63:44	Reserved.	

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2FF0H, 12272	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	
3	Slice 3 select.	
4	Slice 4 select.	
18:5	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: 2FF2H, 12274	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

2.17.6 MSRs Introduced in the Intel® Xeon® Scalable Processor Family

The Intel® Xeon® Scalable Processor Family (CUID Signature DisplayFamily_DisplayModel value of 06_55H) supports the MSRs listed in Table 2-50.

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CUID Signature DisplayFamily_DisplayModel Value of 06_55H

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	
2	Enable VMX Outside SMX Operation (R/WL)	
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
18	SGX Global Functions Enable (R/WL)	
20	LMCE_ENABLED (R/WL)	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:21	Reserved.	
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/WO) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved.	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) See Table 2-26.	Package
22:16	Reserved.	
23	PPIN_CAP (R/O) See Table 2-26.	Package
27:24	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.	Package
30	Programmable TJ OFFSET (R/O) See Table 2-26.	Package
39:31	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) See Table 2-26.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/W0)	
16	Automatic C-State Conversion Enable (R/W) If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6).	
24:17	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count.	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	MCG_EM_P	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and a host-space interface is available to SMM handler.	
59	Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and a host-space interface is available to SMM handler.	
63:60	Reserved.	
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	
7	Thermal Threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WC0) See Table 2-2.	
12	Current Limit Status (R/O) See Table 2-2.	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Current Limit Log (R/WC0) See Table 2-2.	
14	Cross Domain Limit Status (R/O) See Table 2-2.	
15	Cross Domain Limit Log (R/WC0) See Table 2-2.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R/O) See Table 2-26.	
27:24	TCC Activation Offset (R/W) See Table 2-26.	
63:28	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
This register defines the ratio limits. RATIO[0:7] must be populated in ascending order. RATIO[i+1] must be less than or equal to RATIO[i]. Entries with RATIO[i] will be ignored. If any of the rules above are broken, the configuration is silently rejected. If the programmed ratio is:		Package
- Above the fused ratio for that core count, it will be clipped to the fuse limits (assuming !OC). - Below the min supported ratio, it will be clipped.		
7:0	RATIO_0 Defines ratio limits.	
15:8	RATIO_1 Defines ratio limits.	
23:16	RATIO_2 Defines ratio limits.	
31:24	RATIO_3 Defines ratio limits.	
39:32	RATIO_4 Defines ratio limits.	
47:40	RATIO_5 Defines ratio limits.	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
55:48	RATIO_6 Defines ratio limits.	
63:56	RATIO_7 Defines ratio limits.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT_CORES	
This register defines the active core ranges for each frequency point. NUMCORE[0:7] must be populated in ascending order. NUMCORE[i+1] must be greater than NUMCORE[i]. Entries with NUMCORE[i] == 0 will be ignored. The last valid entry must have NUMCORE >= the number of cores in the SKU. If any of the rules above are broken, the configuration is silently rejected.		Package
7:0	NUMCORE_0 Defines the active core ranges for each frequency point.	
15:8	NUMCORE_1 Defines the active core ranges for each frequency point.	
23:16	NUMCORE_2 Defines the active core ranges for each frequency point.	
31:24	NUMCORE_3 Defines the active core ranges for each frequency point.	
39:32	NUMCORE_4 Defines the active core ranges for each frequency point.	
47:40	NUMCORE_5 Defines the active core ranges for each frequency point.	
55:48	NUMCORE_6 Defines the active core ranges for each frequency point.	
63:56	NUMCORE_7 Defines the active core ranges for each frequency point.	
Register Address: 280H, 640	IA32_MCO_CTL2	
See Table 2-2.		Core
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Core
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MCO reports MC errors from the IFU module.		Core
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MCO reports MC errors from the IFU module.		Core
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MCO reports MC errors from the IFU module.		Core
Register Address: 403H, 1027	IA32_MCO_MISC	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC0 reports MC errors from the IFU module.		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 407H, 1031	IA32_MC1_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.	Package	
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.	Package	
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.	Package	
Register Address: 413H, 1043	IA32_MC4_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.	Package	
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.	Package	
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.	Package	
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.	Package	
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.	Package	
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.	Package	
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.	Package	
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.	Package	
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.	Package	
Register Address: 41CH, 1052	IA32_MC7_CTL	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 17.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 17.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
63:15	Reserved.	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
7	Reserved.	
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread
7:0	EventID (R/W) Event encoding: 0x00: No monitoring. 0x01: L3 occupancy monitoring. 0x02: Total memory bandwidth monitoring. 0x03: Local memory bandwidth monitoring. All other encoding reserved.	
31:8	Reserved.	
41:32	RMID (R/W)	
63:42	Reserved.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
9:0	RMID	
31:10	Reserved.	
51:32	CLOS (R/W)	
63: 52	Reserved.	
Register Address: C90H, 3216	IA32_L3_QOS_MASK_0	
L3 Class Of Service Mask - CLOS 0 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 0.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 0 enforcement.	
63:20	Reserved.	
Register Address: C91H, 3217	IA32_L3_QOS_MASK_1	
L3 Class Of Service Mask - CLOS 1 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 1.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 1 enforcement.	
63:20	Reserved.	
Register Address: C92H, 3218	IA32_L3_QOS_MASK_2	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
L3 Class Of Service Mask - CLOS 2 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 2 >= 2.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 2 enforcement.	
63:20	Reserved.	
Register Address: C93H, 3219	IA32_L3_QOS_MASK_3	
L3 Class Of Service Mask - CLOS 3 (R/W). If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 3.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: C94H, 3220	IA32_L3_QOS_MASK_4	
L3 Class Of Service Mask - CLOS 4 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 4.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 4 enforcement.	
63:20	Reserved.	
Register Address: C95H, 3221	IA32_L3_QOS_MASK_5	
L3 Class Of Service Mask - CLOS 5 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 5.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 5 enforcement.	
63:20	Reserved.	
Register Address: C96H, 3222	IA32_L3_QOS_MASK_6	
L3 Class Of Service Mask - CLOS 6 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 6.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 6 enforcement.	
63:20	Reserved.	
Register Address: C97H, 3223	IA32_L3_QOS_MASK_7	
L3 Class Of Service Mask - CLOS 7 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 7.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 7 enforcement.	
63:20	Reserved.	
Register Address: C98H, 3224	IA32_L3_QOS_MASK_8	
L3 Class Of Service Mask - CLOS 8 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 8.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 8 enforcement.	
63:20	Reserved.	
Register Address: C99H, 3225	IA32_L3_QOS_MASK_9	
L3 Class Of Service Mask - CLOS 9 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 9.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 9 enforcement.	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:20	Reserved.	
Register Address: C9AH, 3226	IA32_L3_QOS_MASK_10	
L3 Class Of Service Mask - CLOS 10 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 10.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 10 enforcement.	
63:20	Reserved.	
Register Address: C9BH, 3227	IA32_L3_QOS_MASK_11	
L3 Class Of Service Mask - CLOS 11 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 11.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 11 enforcement.	
63:20	Reserved.	
Register Address: C9CH, 3228	IA32_L3_QOS_MASK_12	
L3 Class Of Service Mask - CLOS 12 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 12.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 12 enforcement.	
63:20	Reserved.	
Register Address: C9DH, 3229	IA32_L3_QOS_MASK_13	
L3 Class Of Service Mask - CLOS 13 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 13.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 13 enforcement.	
63:20	Reserved.	
Register Address: C9EH, 3230	IA32_L3_QOS_MASK_14	
L3 Class Of Service Mask - CLOS 14 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 14.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 14 enforcement.	
63:20	Reserved.	
Register Address: C9FH, 3231	IA32_L3_QOS_MASK_15	
L3 Class Of Service Mask - CLOS 15 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 15.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 15 enforcement.	
63:20	Reserved.	

2.17.7 MSRs Specific to the 3rd Generation Intel® Xeon® Scalable Processor Family Based on Ice Lake Microarchitecture

The 3rd generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_6AH or 06_6CH) support the MSRs listed in Table 2-51.

Table 2-51. MSRs Supported by the 3rd Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_6AH or 06_6CH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 612H, 1554	MSR_PACKAGE_ENERGY_TIME_STATUS	
Package energy consumed by the entire CPU (R/W)		Package
31:0	Total amount of energy consumed since last reset.	
63:32	Total time elapsed when the energy was last updated. This is a monotonic increment counter with auto wrap back to zero after overflow. Unit is 10ns.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
Allows software to set power limits for the DRAM domain and measurement attributes associated with each limit.		Package
14:0	DRAM_PP_PWR_LIM: Power Limit[0] for DDR domain. Units = Watts, Format = 11.3, Resolution = 0.125W, Range = 0-2047.875W.	
15	PWR_LIM_CTRL_EN: Power Limit[0] enable bit for DDR domain.	
16	Reserved.	
23:17	CTRL_TIME_WIN: Power Limit[0] time window Y value, for DDR domain. Actual time_window for RAPL is: $(1/1024 \text{ seconds}) * (1 + (x/4)) * (2^y)$	
62:24	Reserved.	
63	PP_PWR_LIM_LOCK: When set, this entire register becomes read-only. This bit will typically be set by BIOS during boot.	
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM Power Parameters (R/W)		Package
14:0	Spec DRAM Power (DRAM_TDP): The Spec power allowed for DRAM. The TDP setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].	
15	Reserved.	

Table 2-51. MSRs Supported by the 3rd Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_6AH or 06_6CH (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
30:16	Minimal DRAM Power (DRAM_MIN_PWR): The minimal power setting allowed for DRAM. Lower values will be clamped to this value. The minimum setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].	
31	Reserved.	
46:32	Maximal Package Power (DRAM_MAX_PWR): The maximal power setting allowed for DRAM. Higher values will be clamped to this value. The maximum setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].	
47	Reserved.	
54:48	Maximal Time Window (DRAM_MAX_WIN): The maximal time window allowed for the DRAM. Higher values will be clamped to this value. $x = \text{PKG\_MAX\_WIN}[54:53]$ $y = \text{PKG\_MAX\_WIN}[52:48]$ The timing interval window is a floating-point number given by $1.x * \text{power}(2,y)$. The unit of measurement is defined in MSR_DRAM_POWER_INFO_UNIT[TIME_UNIT].	
62:55	Reserved.	
63	LOCK: Lock bit to lock the register.	
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
See Table 2-2.		
Register Address: 982H, 2434	IA32_TME_ACTIVATE	
See Table 2-2.		
Register Address: 983H, 2435	IA32_TME_EXCLUDE_MASK	
See Table 2-2.		
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
See Table 2-2.		

2.17.8 MSRs Specific to the 4th and 5th Generation Intel® Xeon® Scalable Processor Families

The 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_8FH) and the 5th generation Intel® Xeon® Scalable Processor Family based on Emerald Rapids microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_CFH) both support the MSRs listed in Section 2.17, “MSRs In the 6th—13th Generation Intel® Core™ Proces-

sors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 P-core processors, Intel® Xeon® 6 E-core processors, and Intel® Series 2 Core™ Ultra Processors,” including Table 2-52. For an MSR listed in Table 2-52 that also appears in the model-specific tables of prior generations, Table 2-52 supersedes prior generation tables.

Certain bit field positions may be related to the maximum physical address width, the value of which is expressed as “MAXPHYADDR” in Table 2-52. See Section 2.1 for an explanation of this value.

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register (R/W)		Core
27:0	Reserved.	
28	UC_LOCK_DISABLE If set to 1, a UC lock will cause a #GP(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
31:30	Reserved.	
Register Address: A7H, 167	MSR_BIOS_DEBUG	
BIOS DEBUG (R/O) See Table 2-45.		Thread
Register Address: BCH, 188	IA32_MISC_PACKAGE_CTLs	
Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.		Package
Register Address: BFH, 191	IA32_PBOPT_CTRL	
IA32_PBOPT_OPT_CTRL If IA32_ARCH_CAPABILITIES[32] = 1 This MSR provides an architectural enumeration function for model-specific behavior.		
0	PREDICTION_CONTROL_BARRIER (R/W) When set to 0 (default), original Indirect Branch Predictor Barrier Target Array Return (IBPB TA RET) mitigation is enabled. When set to 1, alternative IBPB mitigation is enabled.	
63:1	Reserved.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/W) If CUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Core
0	Reserved: returns zero.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	Reserved: returns zero.	
2	INTEGRITY_CAPABILITIES When set to 1, the processor supports MSR_INTEGRITY_CAPABILITIES.	
3	RSM_IN_CPLD_ONLY Indicates that RSM will only be allowed in CPLD and will #GP for all non-CPLD privilege levels.	
4	UC_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 28 of MSR_MEMORY_CTRL (MSR address 33H).	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL.	
6	Reserved: returns zero.	
7	UC_STORE_THROTTLING_SUPPORTED Indicates that the snoop filter quality of service MSRs are supported on this core. This is based on the existence of a non-inclusive cache and the L2/MLC QoS feature supported.	
63:8	Reserved: returns zero.	
Register Address: E1H, 225	IA32_UWAIT_CONTROL	
UWAIT Control (R/W) See Table 2-2.		
Register Address: EDH, 237	MSR_RAR_CONTROL	
RAR Control (R/W)		Thread
29:0	Reserved.	
30	IGNORE_IF Allow RAR servicing at the RLP regardless of the value of RFLAGS.IF.	
31	ENABLE RAR events are recognized. When RAR is not enabled, RARs are dropped.	
63:32	Reserved.	
Register Address: EEH, 238	MSR_RAR_ACTION_VECTOR_BASE	
Pointer to RAR Action Vector (R/W)		Thread
5:0	Reserved.	
MAXPHYADDR-1:6	VECTOR_PHYSICAL_ADDRESS Pointer to the physical address of the 64B aligned RAR action vector.	
63:MAXPHYADDR	Reserved.	
Register Address: EFH, 239	MSR_RAR_PAYLOAD_TABLE_BASE	
Pointer to Base of RAR Payload Table (R/W)		Thread
11:0	Reserved.	
MAXPHYADDR-1:12	TABLE_PHYSICAL_ADDRESS Pointer to the base physical address of the 4K aligned RAR payload table.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:MAXPHYADDR	Reserved.	
Register Address: F0H, 240	MSR_RAR_INFO	
Read Only RAR Information (R/O)		Thread
31:0	Supported payload type bitmap. A value of 1 in bit position [i] indicates that payload type [i] is supported.	
37:32	Table Max Index Maximum supported payload table index.	
63:38	Always zero.	
Register Address: 105H, 261	MSR_CORE_BIST	
Core BIST (R/W) Controls Array BIST activation and status checking as part of FUSA.		Core
31:0	BIST_ARRAY Bitmap indicating which arrays to run BIST on (WRITE). Bitmap indicating which arrays were not processed, i.e., completion mask (READ).	
39:32	BANK Array bank of the [least significant set bit] array indicated in EAX to start BIST(WRITE). Array bank interrupted or failed (READ).	
47:40	DWORD Array dword of the [least significant set bit] array indicated in EAX to start BIST (WRITE). Array dword interrupted or failed (READ).	
62:48	Reserved.	
63	CTRL_RESULT Indicates whether WRMSR should signal Machine-Check upon BIST-error (WRITE). BIST result PASS(0)/FAIL(1) of the (least significant set bit) array indicated in EAX (READ).	
Register Address: 10AH, 266	IA32_ARCH_CAPABILITIES	
Enumeration of Architectural Features (R/O) See Table 2-2.		
Register Address: 1A4H, 420	MSR_PREFETCH_CONTROL	
Prefetch Disable Bits (R/W)		
0	L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	
1	L2_ADJACENT_CACHE_LINE_PREFETCHER_DISABLE If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	DCU_HARDWARE_PREFETCHER_DISABLE If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	
3	DCU_IP_PREFETCHER_DISABLE If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	
4	Reserved.	
5	AMP_PREFETCH_DISABLE If 1, disables the L2 Adaptive Multipath Probability (AMP) prefetcher.	
63:6	Reserved.	
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	
Primary Maximum Turbo Ratio Limit (R/W) See Table 2-46.		Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT_CORES	
See Table 2-50.		Package
Register Address: 1C4H, 452	IA32_XFD	
Extended Feature Detect (R/W) See Table 2-2.		
Register Address: 1C5H, 453	IA32_XFD_ERR	
XFD Error Code (R/W) See Table 2-2.		
Register Address: 2C2H, 706	MSR_COPY_SCAN_HASHES	
COPY_SCAN_HASHES (W)		Die
63:0	SCAN_HASH_ADDR Contains the linear address of the SCAN Test HASH Binary loaded into memory.	
Register Address: 2C3H, 707	MSR_SCAN_HASHES_STATUS	
SCAN_HASHES_STATUS (R/O)		
15:0	CHUNK_SIZE Chunk size of the test in KB.	Die
23:16	NUM_CHUNKS Total number of chunks.	Die
31:24	Reserved: all zeros.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
39:32	ERROR_CODE The error-code refers to the LP that runs WRMSR(2C2H). 0x0: No error reported. 0x1: Attempt to copy scan-hashes when copy already in progress. 0x2: Secure Memory not set up correctly. 0x3: Scan-image header Image_info.ProgramID doesn't match RDMSR(2D9H)[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. 0x4: Reserved 0x5: Integrity check failed. 0x6: Re-install of scan test image attempted when current scan test image is in use by other LPs.	Thread
50:40	Reserved: set to all zeros.	
62:51	MAX_CORE_LIMIT Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.	Die
63	Valid Valid bit is set when COPY_SCAN_HASHES has completed successfully.	Die
Register Address: 2C4H, 708	MSR_AUTHENTICATE_AND_COPY_CHUNK	
AUTHENTICATE_AND_COPY_CHUNK (W)		Die
7:0	CHUNK_INDEX Chunk Index, should be less than the total number of chunks defined by NUM_CHUNKS (MSR_SCAN_HASHES_STATUS[23:16]).	
63:8	CHUNK_ADDR Bits 63:8 of 256B aligned Linear address of scan chunk in memory.	
Register Address: 2C5H, 709	MSR_CHUNKS_AUTHENTICATION_STATUS	
CHUNKS_AUTHENTICATION_STATUS (R/O)		
7:0	VALID_CHUNKS Total number of Valid (authenticated) chunks.	Die
15:8	TOTAL_CHUNKS Total number of chunks.	Die
31:16	Reserved: all zeros.	
39:32	ERROR_CODE The error code refers to the LP that runs WRMSR(2C4H). 0x0: No error reported. 0x1: Attempt to authenticate a CHUNK which is already marked as authentic or is currently being installed by another core. 0x2: CHUNK authentication error. HASH of chunk did not match expected value.	Thread
63:40	Reserved: set to all zeros.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2C6H, 710	MSR_ACTIVATE_SCAN	
ACTIVATE_SCAN (W)		Thread
7:0	CHUNK_START_INDEX Indicates chunk index to start from.	
15:8	CHUNK_STOP_INDEX Indicates what chunk index to stop at (inclusive).	
31:16	Reserved: all zeros.	
62:32	THREAD_WAIT_DELAY TSC based delay to allow threads to rendezvous.	
63	SIGNAL_MCE If 1, then on scan-error log MC in MC4_STATUS and signal MCE if machine check signaling enabled in MC4_CTL[0]. If 0, then no logging/no signaling.	
Register Address: 2C7H, 711	MSR_SCAN_STATUS	
SCAN_STATUS (R/O)		
7:0	CHUNK_NUM SCAN Chunk that was reached.	Core
15:8	CHUNK_STOP_INDEX Indicates what chunk index to stop at (inclusive). Maps to same field in WRMSR(ACTIVATE_SCAN).	Core
31:16	Reserved: return all zeros.	
39:32	ERROR_CODE 0x0: No error. 0x1: SCAN operation did not start. Other thread did not join in time. 0x2: SCAN operation did not start. Interrupt occurred prior to threads rendezvous. 0x3: SCAN operation did not start. Power Management conditions are inadequate to run Intel In-field Scan. 0x4: SCAN operation did not start. Non-valid chunks in the range CHUNK_STOP_INDEX : CHUNK_START_INDEX. 0x5: SCAN operation did not start. Mismatch in arguments between threads T0/T1. 0x6: SCAN operation did not start. Core not capable of performing SCAN currently. 0x8: SCAN operation did not start. Exceeded number of Logical Processors (LP) allowed to run Intel In-field Scan concurrently. MAX_CORE_LIMIT exceeded. 0x9: Interrupt occurred. Scan operation aborted prematurely, not all chunks requested have been executed.	Thread
61:40	Reserved: return all zeros.	
62	SCAN_CONTROL_ERROR Scan-System-Controller malfunction.	Core

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63	SCAN_SIGNATURE_ERROR Core failed SCAN-SIGNATURE checking for this chunk.	Core
Register Address: 2C8H, 712	MSR_SCAN_MODULE_ID	
SCAN_MODULE_ID (R/O)		Module
31:0	RevID of the currently installed scan test image. Maps to Revision field in external header (offset 4).	
63:32	Reserved: return all zeros.	
Register Address: 2C9H, 713	MSR_LAST_SAF_WP	
LAST_SAF_WP (R/O)		Core
31:0	LAST_WP Provides information about the core when the last WRMSR(ACTIVATE_SCAN) was executed. Available only if enumerated in MSR_INTEGRITY_CAPABILITIES[10:9].	
63:32	Reserved: return all zeros.	
Register Address: 2D9H, 729	MSR_INTEGRITY_CAPABILITIES	
INTEGRITY_CAPABILITIES (R/O)		Module
0	STARTUP_SCAN_BIST When set, supports Intel In-field Scan.	
3:1	Reserved: return all zeros.	
4	PERIODIC_SCAN_BIST When set, supports Intel In-field Scan.	
23:5	Reserved: return all zeros.	
31:24	ID of the scan programs supported for this part. WRMSR(2C2H) verifies this value against the corresponding value in the scan-image header, i.e., Image_info.	
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs," through Section 18.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 413H, 1043	IA32_MC4_MISC	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 18.3.2.1, "IA32_MCI_CTL MSRs," through Section 18.3.2.4, "IA32_MCI_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTL3	
Capability Reporting Register of Tertiary Processor-Based VM-Execution Controls (R/O) See Table 2-2.		
Register Address: 493H, 1171	IA32_VMX_EXIT_CTL2	
Capability Reporting Register of Secondary VM-Exit Controls (R/O) See Table 2-2.		
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W) See Table 2-47.		Thread
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package
31:0	Energy in 61 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 64DH, 1613	MSR_PLATFORM_ENERGY_STATUS	
Platform Energy Status (R/O)		Package
31:0	TOTAL_ENERGY_CONSUMED Total energy consumption in J (32.0), in 10nsec units.	
63:32	TIME_STAMP Time stamp (U32.0).	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
Platform Power Limit Control (R/W-L)		Package
16:0	POWER_LIMIT_1 The average power limit value that the platform must not exceed over a time window as specified by the Power_Limit_1_TIME field. The default value is the Thermal Design Power (TDP) and varies with product skus. The unit is specified in MSR_RAPL_POWER_UNIT.	
17	POWER_LIMIT_1_EN When set, the processor can apply control policies such that the platform average power does not exceed the Power_Limit_1 value over an exponential weighted moving average of the time window.	
18	CRITICAL_POWER_CLAMP_1 When set, the processor can go below the OS-requested P States to maintain the power below the specified Power_Limit_1 value.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
25:19	POWER_LIMIT_1_TIME This indicates the time window over which the Power_Limit_1 value should be maintained. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, and the unit is specified in MSR_RAPL_POWER_UNIT[Time Unit].	
31:26	Reserved.	
48:32	POWER_LIMIT_2 This is the Duration Power limit value that the platform must not exceed. The unit is specified in MSR_RAPL_POWER_UNIT.	
49	Enable Platform Power Limit #2 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #2 over the Short Duration time window.	
50	Platform Clamping Limitation #2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #2 value.	
57:51	POWER_LIMIT_2_TIME This indicates the time window over which the Power_Limit_2 value should be maintained. This field has the same format as the POWER_LIMIT_1_TIME field.	
62:58	Reserved.	
63	LOCK Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 665H, 1637	MSR_PLATFORM_POWER_INFO	
Platform Power Information (R/W)		Package
16:0	MAX_PPL1 Maximum PP L1 value. The unit is specified in MSR_RAPL_POWER_UNIT.	
31:17	MIN_PPL1 Minimum PP L1 value. The unit is specified in MSR_RAPL_POWER_UNIT.	
48:32	MAX_PPL2 Maximum PP L2 value. The unit is specified in MSR_RAPL_POWER_UNIT.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
55:49	MAX_TW Maximum time window. The unit is specified in MSR_RAPL_POWER_UNIT.	
62:56	Reserved.	
63	LOCK Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 666H, 1638	MSR_PLATFORM_RAPL_SOCKET_PERF_STATUS	
Platform RAPL Socket Performance Status (R/O)		Package
31:0	Count of limited performance due to platform RAPL limit.	
Register Address: 6A0H, 1696	IA32_U_CET	
Configure User Mode CET (R/W) See Table 2-2.		
Register Address: 6A2H, 1698	IA32_S_CET	
Configure Supervisor Mode CET (R/W) See Table 2-2.		
Register Address: 6A4H, 1700	IA32_PL0_SSP	
Linear address to be loaded into SSP on transition to privilege level 0. (R/W) See Table 2-2.		
Register Address: 6A5H, 1701	IA32_PL1_SSP	
Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.		
Register Address: 6A6H, 1702	IA32_PL2_SSP	
Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.		
Register Address: 6A7H, 1703	IA32_PL3_SSP	
Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.		
Register Address: 6A8H, 1704	IA32_INTERRUPT_SSP_TABLE_ADDR	
Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) See Table 2-2.		
Register Address: 6E1H, 1761	IA32_PKRS	
Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.		
Register Address: 776H, 1910	IA32_HWP_CTL	
See Table 2-2.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
Memory Encryption Capability MSR See Table 2-2.		

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 985H, 2437	IA32_UINTR_RR	
User Interrupt Request Register (R/W) See Table 2-2.		
Register Address: 986H, 2438	IA32_UINTR_HANDLER	
User Interrupt Handler Address (R/W) See Table 2-2.		
Register Address: 987H, 2439	IA32_UINTR_STACKADJUST	
User Interrupt Stack Adjustment (R/W) See Table 2-2.		
Register Address: 988H, 2440	IA32_UINTR_MISC	
User-Interrupt Target-Table Size and Notification Vector (R/W) See Table 2-2.		
Register Address: 989H, 2441	IA32_UINTR_PD	
User Interrupt PID Address (R/W) See Table 2-2.		
Register Address: 98AH, 2442	IA32_UINTR_TT	
User-Interrupt Target Table (R/W) See Table 2-2.		
Register Address: C70H, 3184	MSR_B1_PMON_EVNT_SELO	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C71H, 3185	MSR_B1_PMON_CTR0	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C72H, 3186	MSR_B1_PMON_EVNT_SEL1	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C73H, 3187	MSR_B1_PMON_CTR1	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C74H, 3188	MSR_B1_PMON_EVNT_SEL2	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C75H, 3189	MSR_B1_PMON_CTR2	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C76H, 3190	MSR_B1_PMON_EVNT_SEL3	
Uncore B-box 1vPerfMon event select MSR.		Package
Register Address: C77H, 3191	MSR_B1_PMON_CTR3	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C82H, 3122	MSR_W_PMON_BOX_OVF_CTRL	
Uncore W-box PerfMon local box overflow control MSR.		Package
Register Address: C8FH, 3215	IA32_PQR_ASSOC	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		
Register Address: C90H—C9EH, 3216—3230	IA32_L3_QOS_MASK_0 through IA32_L3_QOS_MASK_14	
See Table 2-50.		Package
Register Address: D10H—D17H, 3344—3351	IA32_L2_QOS_MASK_[0-7]	
IA32_CR_L2_QOS_MASK_[0-7] If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. See Table 2-2.		Core
Register Address: D93H, 3475	IA32_PASID	
See Table 2-2.		
Register Address: 1200H—121FH, 4608—4639	IA32_LBR_x_INFO	
Last Branch Record Entry X Info Register (R/W) See Table 2-2.		
Register Address: 1406H, 5126	IA32_MCU_CONTROL	
See Table 2-2.		
Register Address: 14CEH, 5326	IA32_LBR_CTL	
Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.		
Register Address: 14CFH, 5327	IA32_LBR_DEPTH	
Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.		
Register Address: 1500H—151FH, 5376—5407	IA32_LBR_x_FROM_IP	
Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.		
Register Address: 1600H—161FH, 5632—5663	IA32_LBR_x_TO_IP	
Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.		

2.17.9 MSRs Introduced in the Intel® Core™ Ultra 7 Processor Supporting Performance Hybrid Architecture

Table 2-53 lists additional MSRs for the Intel Core Ultra 7 processor with a CPUID Signature DisplayFamily_DisplayModel value of 06_AAH. Table 2-54 lists the MSRs unique to the processor P-core. Table 2-55 lists the MSRs unique to the processor E-core.

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor will only allow one in-progress UC store at a time.	
28	UC_LOCK_DISABLE If set to 1, a UC lock will cause a #GP(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 11.1.2.3, “Features to Disable Bus Locks.”	
63:30	Reserved.	
Register Address: 7AH, 122	IA32_FEATURE_ACTIVATION	
Feature Activation (R/W) Implements Feature Activation command. WRMSR to this address activates all ‘activatable’ features on this thread. See Table 2-2.		
Register Address: 80H, 128	MSR_TRACE_HUB_STH ACPIBAR_BASE	
MSR_TRACE_HUB_STH ACPIBAR_BASE (R/W) This register is used by BIOS to program Trace Hub STH base address that will be used by AET messages.		Thread
0	LOCK Lock bit. If set, this MSR cannot be re-written anymore. The lock bit has to be set in order for the AET packets to be directed to Trace Hub MMIO.	
17:1	Reserved.	
45:18	ADDRESS AET target address in Trace Hub MMIO space.	
63:46	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration (R/W)		Core

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3:0	PKG_C_STATE_LIMIT Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings may be supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10	
7:4	MAX_CORE_C_STATE Possible values are: 0000—reserved; 0001—C1; 0010—C3, 0011—C6.	
9:8	Reserved.	
10	IO_MWAIT_REDIRECTION_ENABLE When set, will map IO_read instructions sent to IO registers PMG_IO_BASE_ADDR.PMB0+0/1/2 to MWAIT(C2,3,4) instructions; applies to deepc4 too.	
14:11	Reserved.	
15	CFG_LOCK When set, locks bits 15:0 of this register for further writes, until the next reset occurs.	
24:16	Reserved.	
25	C3_STATE_AUTO_DEMOTION_ENABLE When set, processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	
26	C1_STATE_AUTO_DEMOTION_ENABLE When set, processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	ENABLE_C3_UNDEMOTION Enable Un-Demotion from Demoted C3.	
28	ENABLE_C1_UNDEMOTION Enable Un-Demotion from Demoted C1.	
29	ENABLE_PKGC_AUTODEMOTION Enable Package C-State Auto-Demotion. It enables use of the history of past package C-state depth and residence, as a factor in determining C-State depth.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
30	ENABLE_PKGC_UNDEMOTION Enable Package C-State Un-Demotion. It enables considering cases where demotion was the incorrect decision in determining C-State depth.	
31	TIMED_MWAIT_ENABLE When set, enables Timed MWAIT feature. MWAIT would #GP on attempts to do setup MWAIT timer if this bit is not set.	
63:32	Reserved.	
Register Address: E4H, 228	MSR_IO_CAPTURE_BASE	
IO Capture Base (R/W) Power Management IO Redirection in C-state. See http://biosbits.org .		Core
15:0	LVL_2_BASE_ADDRESS Specifies the base address visible to software for IO redirection. If MSR_PKG_CST_CONFIG_CONTROL.IO_MWAIT_REDIRECTION_ENABLE, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	CST_RANGE Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL.IO_MWAIT_REDIRECTION_ENABLE: 000b—C3 is the max C-State to include. 001b—C6 is the max C-State to include. 010b—C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Feature Configuration (R/W)		Core
0	AESNI_LOCK Once this bit is set, writes to this register will not be allowed.	
1	AESNI_DISABLE This bit disables Advanced Encryption Standard feature on this processor core. To disable AES, BIOS will write '11 to this MSR on every core.	
63:2	Reserved.	
Register Address: 140H, 320	MSR_FEATURE_ENABLES	
Feature Enable (R/W) Miscellaneous enables for thread specific features.		Thread
0	CPUID_GP_ON_CPL_GT_0 Causes CPUID to #GP if CPL greater than 0 and not in SMM.	
63:1	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target (R/W) Legacy register holding temperature related constants for Platform use.		Package
6:0	TCC Offset Time Window Describes the RATL averaging time window.	
7	TCC Offset Clamping Bit When enabled will allow RATL throttling below P1.	
15:8	Temperature Control Offset Fan Temperature Target Offset (a.k.a. T-Control) indicates the relative offset from the Thermal Monitor Trip Temperature at which fans should be engaged.	
23:16	TCC Activation Temperature The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
30:24	TCC Activation Offset Specifies a temperature offset in degrees C from the temperature target (bits 23:16). PROCHOT# will assert at the offset target temperature. Write is permitted only if MSR_PLATFORM_INFO[30] is set.	
31	LOCKED When set, this entire register becomes read-only.	
63:2	Reserved.	
Register Address: 1A4H, 420	MSR_PREFETCH_CONTROL	
PREFETCH Control (R/W) Prefetch disable bits.		Thread
0	L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	
1	L2_ADJACENT_CACHE_LINE_PREFETCHER_DISABLE If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	
2	DCU_HARDWARE_PREFETCHER_DISABLE If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	
3	DCU_IP_PREFETCHER_DISABLE If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	
4	DCU_NEXT_PAGE_PREFETCH_DISABLE If 1, disables Next Page prefetcher.	
5	AMP_PREFETCH_DISABLE If 1, disables L2 Adaptive Multipath Probability (AMP) prefetcher.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
6	LLC_PAGE_PREFETCH_DISABLE If 1, disables the LLC Page prefetcher.	
7	AOP_PREFETCH_DISABLE	
8	STREAM_PREFETCH_CODE_FETCH_DISABLE	
63:9	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
OFFCORE_RSP_0 (R/W) Offcore Response Event Select Register		Thread
0	TRUE_DEMAND_CACHE_LOAD Demand Data Rd = DCU reads (includes partials) that is not tagged homeless.	
1	DEMAND_RFO Demand Instruction fetch = IFU Fetches. ItoM or RFO that is not tagged homeless.	
2	DEMAND_CODE_READ Demand Instruction fetch = IFU Fetches. CRd or CRd_UC.	
3	CORE_MODIFIED_WRITEBACK WBMtoI or WBMtoE.	
4	HW_PREFETCH_MLC_LOAD L2 prefetcher requests triggered by reads from MEC (except those triggered by I-side).	
5	HW_PREFETCH_MLC_RFO L2 prefetcher requests triggered by RFOs.	
6	HW_PREFETCH_MLC_CODE L2 prefetcher requests triggered by I-side requests.	
7	HW_PREFETCH_LLC_LOAD LLC prefetch requests triggered by DRd.	
8	HW_PREFETCH_LLC_RFO LLC prefetch requests triggered by RFO.	
9	HW_PREFETCH_LLC_CODE LLC prefetch requests triggered by CRd.	
10	L1_HWPREFETCH Covers Hardware PFRFO, PFNEAR, PFMED, PFFAR, PFHW, PFNTA, PFNPP, PFIPP including the homeless versions.	
11	ALL_STREAMING_STORE Write Combining. WCIL or WCILF.	
12	CORE_NON_MODIFIED_WB WBEFtoI or WBEFtoE.	
13	LLC_PREFETCH LLC prefetch of load/code/RFO.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
14	L1_SWPREFETCH Covers Software PFRFO, PFNEAR, PFMED, PFFAR, PFHW, PFNTA, PFNPP, PFIPP including the homeless versions.	
15	OTHER Includes CLFlush, CLFlushOPT, CLDemote, CLWB, Enqueue SetMonitor, PortIn, IntA, Lock, SplitLock, Unlock, SpCyc, ClrMonitor, PortOut, IntPriUp, IntLog, IntPhy, EOI, RdCurr, WbStol, LLCWBInv, LLCInv, NOP, PCOMMIT.	
16	ANY_RESP Match on any response.	
17	SUPPLIER_NONE No Supplier Details. DATA_PRE [6:3] = 0.	
18	LLC_HIT_M_STATE LLC/L3, M-state, DATA_PRE [6:3] = 2.	
19	LLC_HIT_E_STATE LLC/L3, E-state, DATA_PRE [6:3] = 4.	
20	LLC_HIT_S_STATE LLC/L3, S-state, DATA_PRE [6:3] = 6.	
21	LLC_HIT_F_STATE LLC/L3, F-state, DATA_PRE [6:3] = 8.	
22	FAR_MEM_LOCAL Far Memory, Local, DATA_PRE [6:3] = 1.	
23	FAR_MEM_REMOTE_0_HOP Far Memory, Remote 0-hop, DATA_PRE [6:3] = 3.	
24	FAR_MEM_REMOTE_1_HOP Far Memory, Remote 1-hop, DATA_PRE [6:3] = 5.	
25	FAR_MEM_REMOTE_2_PLUS_HOP Far Memory, Rem 2+ hop, DATA_PRE [6:3] = 7.	
26	NEAR_MEM_MISS_LOCAL_NODE LLC Miss Local Node. Near Memory, Local DATA_PRE [6:3] = E.	
27	NEAR_MEM_REMOTE_0_HOP Near Memory, Remote 0-hop, DATA_PRE [6:3] = B	
28	NEAR_MEM_REMOTE_1_HOP Near Memory, Remote 1-hop, DATA_PRE [6:3] = D.	
29	NEAR_MEM_REMOTE_2_PLUS_HOP Near Memory, Remote 2+ hop, DATA_PRE [6:3] = F.	
30	SPL_HIT Snoop Info: SPL-hit, DATA_PRE [2:0] = 6.	
31	SNOOP_NONE No details as to Snoop-related info. Snoop Info: None, DATA_PRE [2:0] = 0.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
32	NOT_NEEDED No snoop was needed to satisfy the request. Snoop Info: Not needed, DATA_PRE [2:0] = 1.	
33	MISS No snoop was needed to satisfy the request. Snoop Info: Miss, DATA_PRE [2:0] = 2.	
34	HIT_NO_FWD A snoop was needed and it Hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. Snoop Info: Hit No Fwd, DATA_PRE [2:0] = 3.	
35	HIT_EF_WITH_FWD A snoop was needed and data was Forwarded from a remote socket. Snoop Info: Hit EF w/Fwd, DATA_PRE [2:0] = 4.	
36	HITM A snoop was needed and it HitMed in local or remote cache. HitM denotes a cache-line was modified before snoop effect. Snoop Info: HitM, DATA_PRE [2:0] = 5.	
37	NON_DRAM Target was non-DRAM system address. Snoop Info: HitM, DATA_PRE [2:0] = 5.	
38	GO_ERR GO-ERR, RspData[3:0] = 0100.	
39	GO_NO_GO GO-NoGO, RspData[3:0] = 0111.	
40	INPKG_MEM_LOCAL In-package Memory, Local, DATA_PRE [6:3] = 9.	
41	INPKG_MEM_NONLOCAL In-package Memory, Non-Local, DATA_PRE [6:3] = C.	
43:42	Reserved.	
44	UC_LOAD PRd or UCRdF.	
45	UC_STORE WiL.	
46	PARTIAL_STREAMING_STORES WCiL.	
47	FULL_STREAMING_STORES WCiLF.	
48	L1_MODIFIED_WB EVICTIION EXTTYPE from MEC.	
49	L2_MODIFIED_WB WBMtol or WBMtoE.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
50	PSMI MemPushWr_NS (PSMI only).	
51	ITOM ItoM.	
63:52	Reserved.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
OFFCORE_RSP_1 (R/W) Offcore Response Event Select Register. See MSR_OFFCORE_RSP_0 (at 1A6H).		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control (R/W) Various model-specific features enumeration. See http://biosbits.org .		Package
0	Reserved.	
1	ENABLE_HWP_VOTING_RIGHT When set (1), The CPU will take into account thread HWP requests for threads that have voting rights only (ignores thread requests if they do not have voting rights). When reset(0), The CPU will take into account all thread HWP requests, even for threads that don't have voting rights. Setting this bit will cause the HWP Base feature bit to be reported in CPUID as present; clearing will cause it to be reported as non-present.	
5:2	Reserved.	
6	ENABLE_HWP Setting this bit will cause the HWP Base feature bit to report as present in CPUID; clearing this bit will cause CPUID to report the feature as non-present.	
7	ENABLE_HWP_INTERRUPT Setting this bit will cause the HWP Interrupt feature CPUID.06H:EAX[8] bit to report as present; clearing will report as non-present.	
8	ENABLE_OUT_OF_BAND_AUTONOMOUS Setting this bit will cause the HWP Autonomous feature bit to report as present; clearing will report as non-present.	
11:9	Reserved.	
12	ENABLE_HWP_EPP Enable HWP EPP. Setting this bit (1) will cause the HWP CPUID.06H:EAX[10] Energy Performance Preference bit to report as present (1); clearing will report as non-present (0).	
13	LOCK Setting this bit will prevent the BIOS specific bits from changing until the next reset. i.e., only Bits [0,22] which are meant for OS use can be changed once the LOCK bit is set.	
63:14	Reserved.	
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Primary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 1F1H, 497	MSR_CRASHLOG_CONTROL	
Crash Log Control (R/W) Write data to a Crash Log configuration.		Thread
0	CDDIS CrashDump_Disable: If set, indicates that Crash Dump is disabled.	
1	EN_GPRS Collect GPRs on a crash dump. Only meaningful when CDDIS is zero.	
2	EN_GPRS_IN_SMM Collect GPRs in SMM on a crash dump. Only meaningful when CDDIS is zero. EN_GPRS will override this control,	
3	TRIPLE_FAULT_SHUTDOWN Collect a crash log on a triple fault shutdown. Only meaningful when CDDIS is zero.	
63:4	Reserved.	
Register Address: 1F5H, 501	MSR_PRMRR_PHYS_MASK	
Processor Reserved Memory Range Register - Physical Mask (R/W)		Core
9:0	Reserved.	
10	LOCK Once set, this bit prevents software from modifying the PRMRR.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
11	VALID This bit serves as the enable for the PRMRR; the PRMRR must be LOCKed before it can be enabled.	
19:12	Reserved.	
45:20	MASK PRMRR Address Mask.	
63:46	Reserved.	
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register (R/W) See http://biosbits.org .		Package
0	ENABLE_BIDIR_PROCHOT Used to enable or disable the response to PROCHOT# input. When set/enabled, the platform can force the CPU to throttle to a lower power condition such as Pn/Pm by asserting proshot#. When clear/disabled (default), the CPU ignores the status of the proshot input signal.	
1	C1E_ENABLE When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	
2	SAPM_IMC_C2_POLICY This bit determines if self-refresh activation is allowed when entering Package C2 State. If it is set to 0b, PCODE will keep the FORCE_SR_OFF bit asserted in Package C2 State and allow its negation according to the defined latency negotiations with the PCH and Display Engine in Package C3 and deeper states. Otherwise, self-refresh is allowed in Package C2 State.	
3	FAST_BRK_SNP_EN This bit controls the VID swing rate for the OTHER_SNP_WAKE events that are detected by the iMPH. This is the event that is detected by the iMPH when a non-DMI snoopable request is observed while UCLK domain is not functional. 0b: Use slow VID swing rate. 1b: Use fast VID swing rate.	
17:4	Reserved.	
18	PWR_PERF_PLTFRM_OVR Power performance platform override.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
19	EE_TURBO_DISABLE Setting this bit disables the P-States energy efficiency optimization. Default value is 0. Disable/enable the energy efficiency optimization in P-State legacy mode (when IA32_PM_ENABLE[HWP_ENABLE] = 0), has an effect only in the turbo range or into PERF_MIN_CTL value if it is not zero set. In HWP mode (IA32_PM_ENABLE[HWP_ENABLE] == 1), has an effect between the OS desired or OS maximize to the OS minimize performance setting.	
20	RTH_DISABLE Setting this bit disables the Race to Halt optimization and avoids this optimization limitation to execute below the most efficient frequency ratio. Default value is 0 for processors that support Race to Halt optimization.	
21	DIS_PROCHOT_OUT Prochot output disable.	
22	PROCHOT_RESPONSE Prochhot configurable response enable.	
23	VR_THERM_ALERT_DISABLE_LOCK When set to 1, locks PROCHOT related bits of this MSR. Once set, a reset is required to clear this bit.	
24	VR_THERM_ALERT_DISABLE When set to 1, disables the VR_THERMAL_ALERT signaling.	
25	DISABLE_RING_EE Disable Ring EE.	
26	DISABLE_SA_OPTIMIZATION Disable SA optimization.	
27	DISABLE_OOK Disable OOK.	
28	DISABLE_AUTONOMOUS Disable HWP autonomous mode.	
29	Reserved.	
30	CSTATE_PREWAKE_DISABLE C-state pre-wake disable.	
63:31	Reserved.	
Register Address: 2A0H, 672	MSR_PRMRR_BASE_0	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
19:4	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
45:20	BASE PRMRR base address.	
63:46	Reserved.	
Register Address: 474H, 1140	IA32_MC29_CTL	
MC29_CTL. See Table 2-2.		Package
Register Address: 475H, 1141	IA32_MC29_STATUS	
MC29_STATUS. See Table 2-2.		Package
Register Address: 476H, 1142	IA32_MC29_ADDR	
MC29_ADDR. See Table 2-2.		Package
Register Address: 477H, 1143	IA32_MC29_MISC	
MC29_MISC. See Table 2-2.		Package
Register Address: 478H, 1144	IA32_MC30_CTL	
MC30_CTL. See Table 2-2.		Package
Register Address: 479H, 1145	IA32_MC30_STATUS	
MC30_STATUS. See Table 2-2.		Package
Register Address: 47AH, 1146	IA32_MC30_ADDR	
MC30_ADDR. See Table 2-2.		Package
Register Address: 47BH, 1147	IA32_MC30_MISC	
MC30_MISC. See Table 2-2.		Package
Register Address: 47CH, 1148	IA32_MC31_CTL	
MC31_CTL. See Table 2-2.		Package
Register Address: 47DH, 1149	IA32_MC31_STATUS	
MC31_STATUS. See Table 2-2.		Package
Register Address: 47EH, 1150	IA32_MC31_ADDR	
MC31_ADDR. See Table 2-2.		Package
Register Address: 47FH, 1151	IA32_MC31_MISC	
MC31_MISC. See Table 2-2.		Package
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (R/W) Reports SMM capability enhancement.		Package
0	LOCK When set, locks this register from further changes.	
1	SMM_CPU_SAVE_EN If 0, SMI/RSM will save/restore state in SMRAM If 1, SMI/RSM will save/restore state from SRAM.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	SMM_CODE_CHK_EN When clear (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set, any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.	
63:3	Reserved.	
Register Address: 601H, 1537	MSR_VR_CURRENT_CONFIG	
Power Limit 4 (PL4) (R/W) Package-level maximum power limit (in Watts). It is a proactive, instantaneous limit.		Package
15:0	CURRENT_LIMIT PL4 Value in 0.125 A increments. This field is locked by MSR_VR_CURRENT_CONFIG.LOCK. When the LOCK bit is set to 1, this field becomes Read Only.	
30:16	Reserved.	
31	LOCK This bit will lock the CURRENT_LIMIT settings in this register and will also lock this setting. This means that once set to 1, the CURRENT_LIMIT setting and this bit become Read Only until the next Warm Reset.	
63:32	Reserved.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Min/Max Ratio Limits for Uncore LLC and Ring.		Package
6:0	MAX_CLR_RATIO Maximum allowed ratio for the Ring and Last Level Cache (LLC).	
7	Reserved.	
14:8	MIN_CLR_RATIO Minimum allowed ratio for the Ring and Last Level Cache (LLC).	
63:15	Reserved.	
Register Address: 638H, 1592	MSR_PPO_POWER_LIMIT	
MSR_PPO_POWER_LIMIT (R/W) PPO RAPL power unit control.		Package
14:0	IA_PP_PWR_LIM This is the power limitation on the IA cores power plane. The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSR[PWR_UNIT].	
15	PWR_LIM_CTRL_EN This bit must be set in order to limit the power of the IA cores power plane. 0b: IA cores power plane power limitation is disabled. 1b: IA cores power plane power limitation is enabled.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
16	PP_CLAMP_LIM Power Plane Clamping limitation; allow going below P1. 0b: PBM is limited between P1 and P0. 1b: PBM can go below P1.	
23:17	CTRL_TIME_WIN x = CTRL_TIME_WIN[23:22] y = CTRL_TIME_WIN[21:17] The timing interval window is Floating Point number given by $1.x * \text{power}(2,y)$. The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSR[TIME_UNIT]. The maximal time window is bounded by PACKAGE_POWER_SKU_MSR[PKG_MAX_WIN]. The minimum time window is 1 unit of measurement (as defined above).	
30:24	Reserved.	
31	PP_PWR_LIM_LOCK When set, all settings in this register are locked and are treated as Read Only.	
63:32	Reserved.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
Core Performance Limit Reasons Indicator of Frequency Clipping in Processor Cores. (Frequency refers to processor core frequency.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	RSR_LIMIT (R/O) Residency State Regulation Status. When set, frequency is reduced below the operating system request due to residency state regulation limit.	
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced below the operating system request due to Running Average Thermal Limit (RATL).	
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced below the operating system request due to a thermal alert from a processor Voltage Regulator (VR).	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
7	VR_TDC (R/O) VR Therm Design Current Status. When set, frequency is reduced below the operating system request due to VR thermal design current limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced below the operating system request due to electrical or other constraints.	
9	Reserved.	
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL2/PL3.	
12	MAX_TURBO_LIMIT (R/O) Max Turbo Limit Status. When set, frequency is reduced below the operating system request due to multi-core turbo limits.	
13	TURBO_ATTEN (R/O) Turbo Transition Attenuation Status. When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
15:14	Reserved.	
16	PROCHOT_LOG (R/W) PROCHOT Log. When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	THERMAL_LOG (R/W) Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	RSR_LIMIT_LOG (R/W) Residency State Regulation Log. When set, indicates that the Residency State Regulation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	RATL_LOG (R/W) Running average thermal limit Log, RW, When set by PCODE indicates that Running average thermal limit has cause IA frequency clipping. Software should write to this bit to clear the status in this bit.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
22	VR_THERMALERT_LOG (R/W) VR Therm Alert Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR_TDC_LOG (R/W) VR Thermal Design Current Log. When set, indicates that the VR TDC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	OTHER_LOG (R/W) Other Log. When set, indicates that the Other Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	PBM_PL1_LOG (R/W) Package/Platform-Level PL1 Power Limiting Log. When set, indicates that the Package or Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	PBM_PL2_LOG (R/W) Package/Platform-Level PL2 Power Limiting Log. When set, indicates that the Package or Platform Level PL2/PL3 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	MAX_TURBO_LIMIT_LOG (R/W) Max Turbo Limit Log. When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	TURBO_ATTEN_LOG (R/W) Turbo Transition Attenuation Log. When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 650H, 1616	MSR_SECONDARY_TURBO_RATIO_LIMIT	
Secondary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
Platform Power Limit Control (R/W) Allows platform BIOS to limit power consumption of the platform devices to the specified values. The Long Duration power consumption is specified via Platform_Power_Limit_1 and Platform_Power_Limit_1_Time. The Short Duration power consumption limit is specified via the Platform_Power_Limit_2 with duration chosen by the processor. The processor implements an exponential-weighted algorithm in the placement of the time windows.		Package
14:0	POWER_LIMIT_1 Average Power limit value which the platform must not exceed over a time window as specified by Power_Limit_1_TIME field. The default value is the Thermal Design Power (a.k.a TDP) and varies with product skus. The unit is specified in MSR_RAPLPOWER_UNIT.	
15	POWER_LIMIT_1_EN When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit 1 over the time window specified by Power Limit 1 Time Window.	
16	CRITICAL_POWER_CLAMP_1 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit 1 value.	
23:17	POWER_LIMIT_1_TIME Specifies the duration of the time window over which Platform Power Limit 1 value should be maintained for sustained long duration. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, The unit is specified in MSR_RAPLPOWER_UNIT[Time Unit]	
31:24	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
46:32	POWER_LIMIT_2 Average Power limit value which the platform must not exceed over the Short Duration time window chosen by the processor. The recommended default value is 1.25 times the Long Duration Power Limit (i.e., Platform Power Limit 1).	
47	POWER_LIMIT_2_EN When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit 2 over the Short Duration time window.	
48	CRITICAL_POWER_CLAMP_2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit 2 value.	
62:49	Reserved.	
63	LOCK Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 6BOH, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
MSR_GRAPHICS_PERF_LIMIT_REASONS Indicator of Frequency Clipping in the Processor Graphics. (Frequency refers to processor graphics frequency.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced due to running average thermal limit.	
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR_TDC (R/O) VR Thermal Design Current Status. When set, frequency is reduced due to VR TDC limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
12	INEFFICIENT_OPERATION (R/O) Inefficient Operation Status. When set, processor graphics frequency is operating below target frequency.	
15:13	Reserved.	
16	PROCHOT_LOG (R/W) PROCHOT Log. When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	THERMAL_LOG (R/W) Thermal Log. When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	
21	RATL_LOG (R/W) Running Average Thermal Limit Log. When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR_THERMALERT_LOG (R/W) VR Therm Alert Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR_TDC_LOG (R/W) VR Thermal Design Current Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	OTHER_LOG (R/W) Other Log. When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	PBM_PL1_LOG (R/W) Package/Platform-Level PL1 Power Limiting Log. When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
27	PBM_PL2_LOG (R/W) Package/Platform-Level PL2 Power Limiting Log. When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	INEFFICIENT_OPERATION_LOG (R/W) Inefficient Operation Log. When set, indicates that the Inefficient Operation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:29	Reserved.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
MSR_RING_PERF_LIMIT_REASONS Indicator of Frequency Clipping in the Ring Interconnect. (Frequency refers to ring interconnect in the uncore.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced due to running average thermal limit.	
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR_TDC (R/O) VR Thermal Design Current Status. When set, frequency is reduced due to VR TDC limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
15:12	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
16	PROCHOT_LOG (R/W) PROCHOT Log. When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	THERMAL_LOG (R/W) Thermal Log. When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	
21	RATL_LOG (R/W) Running Average Thermal Limit Log. When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR_THERMALERT_LOG (R/W) VR Therm Alert Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR_TDC_LOG (R/W) VR Thermal Design Current Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	OTHER_LOG (R/W) Other Log. When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	PBM_PL1_LOG (R/W) Package/Platform-Level PL1 Power Limiting Log. When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	PBM_PL2_LOG (R/W) Package/Platform-Level PL2 Power Limiting Log. When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:28	Reserved.	
Register Address: 9FBH, 2555	IA32_TME_CLEAR_SAVED_KEY	
IA32_TME_CLEAR_SAVED_KEY (R/W) See Table 2-2.		Package
Register Address: 9FFH, 2559	MSR_CORE_MKTME_ACTIVATE	
MSR_CORE_MKTME_ACTIVATE (R/O) MSR to read TME_ACTIVATE[MK_TME_KEYID_BITS].		Core

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:0	Reserved.	
35:32	READ_MK_TME_KEYID_BITS This value will be returned on a RDMSR, but must be zero on a WRMSR.	
39:36	TDX_RESERVED_KEYID_BITS (read only) The number of key identifier bits allocated to TDX usage. This is a read-only field and must be zero on a WRMSR.	
63:40	Reserved.	

The MSRs listed in Table 2-54 are unique to the Intel Core Ultra 7 processor P-core. These MSRs are not supported on the processor E-core.

Table 2-54. MSRs Supported by the Intel® Core™ Ultra 7 Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 30CH, 780	IA32_FIXED_CTR3	
Fixed-Function Performance Counter 3 (R/W)		Thread
47:0	FIXED_COUNTER Top-down Microarchitecture Analysis unhalting number of available slots counter.	
63:48	Reserved.	
Register Address: 329H, 809	MSR_PERF_METRICS	
Performance Metrics (R/W) This register provides built-in support for Top-down Micro-architecture Analysis (TMA) metrics. It exposes the four TMA Level 1 metrics where the lower 32 bits are divided into four 8 bit fields, each of which is an integer percentage of the total TOPDOWN.SLOTS (as reported by fixed counter 3).		Thread
7:0	RETIRING Percent of utilized by uops that eventually retire (commit).	
15:8	BAD_SPECULATION Percent of Wasted due to incorrect speculation, covering Utilized by uops that do not retire, or Recovery Bubbles (unutilized slots).	
23:16	FRONTEND_BOUND Percent of Unutilized slots where Front-end did not deliver a uop while Back-end is ready.	
31:24	BACKEND_BOUND Percent of Unutilized slots where a uop was not delivered to Back-end due to lack of Back-end resources.	
39:32	MULTI_UOPS Frontend bound.	
47:40	BRANCH_MISPREDICTS Frontend bound.	

Table 2-54. MSRs Supported by the Intel® Core™ Ultra 7 Processor P-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
55:48	FRONTEND_LATENCY Frontend bound.	
63:56	MEMORY_BOUND Frontend bound.	
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W) See Table 2-47.		Thread
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W) See Table 2-44.		Core

The MSRs listed in Table 2-48 are unique to the Intel Core Ultra 7 processor E-core. These MSRs are not supported on the processor P-core.

Table 2-55. MSRs Supported by the Intel® Core™ Ultra 7 Processor E-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4F0H, 1264	MSR_SAF_CTRL	
SAF Control (W/O) Extension to SAF.		Package
0	INVALIDATE_CURRENT_STRIDE Invalidate all chunks in current stride.	
63:1	Reserved.	
Register Address: D18H–D1FH, 3352–3359	IA32_L2_MASK_[8-15]	
IA32_L2_MASK_[8-15] (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Module
15:0	WAY_MASK Capacity Bit Mask. Available ways vectors for class of service of IA core. ‘1 in bit indicates allocation to the way is allowed. ‘0 indicates allocation to the way is not allowed.	
31:16	Reserved.	
Register Address: 1309H–130BH, 4873–4875	MSR_RELOAD_FIXED_CTRx	
Reload value for IA32_FIXED_CTRx (R/W)		Thread
47:0	Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBs_ENABLE, and FIXED_CTRx is overflowed.	
63:48	Reserved.	
Register Address: 14C1H–14C8H, 5313 –5320	MSR_RELOAD_PMCx	
Reload value for IA32_PMCx (R/W)		Thread

Table 2-55. MSRs Supported by the Intel® Core™ Ultra 7 Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
47:0	Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and PMCx is overflowed.	
63:48	Reserved.	
Register Address: 1A8EH, 6798	MSR_STLB_FILL_TRANSLATION	
STLB Fill Translation (W/O) STLB QoS MSR to fill translations into STLB.		Core
3:0	CLOS Class of service to use for the fill.	
9:4	Reserved.	
10	X Set to 1 when LA is to an executable page.	
11	RW Set to 1 when LA is to a writeable page.	
63:12	LA Logical address to use for fill.	

2.17.10 MSRs Introduced in the Intel® Xeon® 6 P-Core Processors

Table 2-56 lists additional MSRs for the Intel Xeon 6 P-core processors. Intel Xeon 6 P-core processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_ADH or 06_AEH.

For an MSR listed in Table 2-56 that also appears in the model-specific tables of prior generations, Table 2-56 supersedes prior generation tables.

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CONTROL	
MSR_MEMORY_CONTROL (R/W) Disables split locks, which are locked instructions that split a cache line.		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor allows one in-progress, post-retirement UC stores at a time.	
28	UC_LOCK_DISABLE If set to 1, a UC load lock will trigger a fault. If clear to 0, UC load locks proceed normally.	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will trigger an #AC fault. If clear to 0, split locks proceed normally	
63:30	Reserved.	
Register Address: 34H, 52	MSR_SMI_COUNT	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
SMI Counter (R/W)		Thread
31:0	SMI_COUNT Running count of SMI events since the last reset.	
63:32	Reserved.	
Register Address: 39H, 57	MSR_SOCKET_ID	
Socket ID (R/W) Reassigns the package-specific portions of the APIC ID. This MSR is used on scalable DP and high-end MP platforms to resolve legacy-mode APIC ID conflicts.		Package
10:0	PACKAGE_ID: Holds package ID. This reflects the upper bits of the APIC ID.	
63:11	Reserved.	
Register Address: 7AH, 122	IA32_FEATURE_ACTIVATION	
IA32_FEATURE_ACTIVATION (R/W) Implements Feature Activation command. WRMSR to this address activates all 'activatable' features on this thread. See Table 2-2.		Thread
Register Address: 7BH, 123	IA32_MCU_ENUMERATION	
IA32_MCU_ENUMERATION (R/O) Enumeration of architectural features. See Table 2-2.		Package
Register Address: 7CH, 124	IA32_MCU_STATUS	
IA32_MCU_STATUS (R/O) Communicates results from the previous patch loads. See Table 2-2.		Package
Register Address: 82H, 130	IA32_FZM_RANGE_INDEX	
IA32_FZM_RANGE_INDEX (R/W) Index and Domain handle for a valid FZM region. Programmed by SW and used by other FRM MSRs FZM Range Index register to R/W Domain Index. See Table 2-2.		Thread
Register Address: 83H, 131	IA32_FZM_DOMAIN_CONFIG	
IA32_FZM_DOMAIN_CONFIG (R/O) Bit mask of valid regions within the domain identified by FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 84H, 132	IA32_FZM_RANGE_STARTADDR	
IA32_FZM_RANGE_STARTADDR (R/O) Start address of the FZM range pointed to by FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 85H, 133	IA32_FZM_RANGE_ENDADDR	
IA32_FZM_RANGE_ENDADDR (R/O) End address of the specified domain in FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 86H, 134	IA32_FZM_RANGE_WRITESTATUS	
IA32_FZM_RANGE_WRITESTATUS (R/O) Write status of the FZM range pointed to by FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 87H, 135	IA32_MKTME_KEYID_PARTITIONING	
MKTME KEY ID Partitioning (R/O) Enumerates the number of activated KeyIDs for Intel TME-MK and Intel TDX. See Table 2-2.		Package

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 90H, 144	IA32_SGXLEPUBKEYHASH4	
IA32_SGXLEPUBKEYHASH4 (R/W) See Table 2-2.		Thread
Register Address: 91H, 145	IA32_SGXLEPUBKEYHASH5	
IA32_SGXLEPUBKEYHASH5 (R/W) See Table 2-2.		Thread
Register Address: 98H, 152	MSR_SEAM_WBINVDP	
SEAM WBINVDP (R/W) Allows software to WBINVD sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 99H, 153	MSR_SEAM_WBNOINVDP	
SEAM WBNOINVDP (R/W) Allows software to WBNOINVDP sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 9AH, 154	MSR_SEAM_INTR_PENDING	
SEAM Interrupt Pending (R/O) Report out some event pending bits.		Thread
0	INTR Interrupt is pending.	
1	NMI NMI is pending.	
2	SMI SMI is pending.	
4:3	OTHER_EVENTS Other events pending.	
63:5	Reserved.	
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL	
SMM Monitor Control (R/W) The SMM Monitor Configuration involves SMM code specifying the MSEG location and enabling dual-monitor treatment by writing to the corresponding MSR. See Table 2-2.		Thread
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/W) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Core
0	STLB_QOS When set to 1, processor supports STLB QoS.	
1	Reserved.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	INTEGRITY_SUPPORTED When set to 1, processor supports Functional Safety. Specific FUSA capabilities are enumerated in MSR_FUSA_CAPABILITIES.	
3	RSM_IN_CPLO_ONLY Intel System Resources Defense: When set to 1, RSM will only be allowed in CPLO and will #GP for all non-CPLO privilege levels.	
4	UC_LOCK_DISABLE When set to 1, processor supports UC load lock disable.	
5	SPLIT_LOCK_DISABLE When set to 1, processor supports #AC on split locks.	
6	SNP_FILTER_QOS When set to 1, processor supports Snoop Filter Quality of Service MSRs.	
7	UC_STORE_THROTTLING When set to 1, processor supports UC store throttling through MSR_MEMORY_CTRL[UC_STORE_THROTTLE].	
63:8	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R/O) See Table 2-2.		Core
Register Address: 105H, 261	MSR_ARRAY_BIST	
MSR_ARRAY_BIST (R/W) Triggered by writing and reading an MSR that can be written by Ring 0 software.		Core

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:0	ARRAY_LIST: Bit map which indicates which arrays to run MarchC- BIST ▪ Bit[0] MLC Data ▪ Bit[1] MLC Tag ▪ Bit[2] C6SRAM Data (NOP for WRMSR – used for reporting error only) ▪ Bit[3] PMA BIST (NOP for WRMSR – used for reporting error only) ▪ Bit[4] STLB Data ▪ Bit[5] IFU Data ▪ Bit[6] STLB Tag ▪ Bit[7] DCU Data ▪ Bit[8] DSB Data ▪ Bit[9] TMUL Data ▪ Bit[10] UROM pointer0 ▪ Bit[11] UROM pointer1-3 ▪ Bit[12] UROM pointer4-7 ▪ Bit[13] UROM unique0 ▪ Bit[14] UROM unique1/2 The WRMSR will run PBIST on all the arrays indicated in the bitmap, starting from the LSB. NOTE2: C6SRAM[Bit 2] and PMA[Bit 3] are only for reporting and do not execute BIST (done by EDX[15:0]uCode during Fusa-Reset).	
46:32	Reserved.	
62:47	Reserved.	
63	SIGNAL_MCE: Signal MCERR upon BIST failure.	
Register Address: 105H, 261	MSR_ARRAY_BIST_STATUS	
MSR_ARRAY_BIST_STATUS (R/O)		Core
31:0	ARRAY_COMPLETION_MASK Bitmap indicating which arrays from the ARRAY_BIST.ARRAY_LIST was not processed. 1 means not tested and 0 means tested.	
62:32	Reserved. Returns all 0s.	
63	PASS_FAIL: 0 means Pass on all arrays in the WRMSR(ARRAY_BIST.ARRAY_LIST) 1 means Fail on the LSB array in the RDMSR(ARRAY_BIST_STATUS.ARRAY_COMPLETION_MASK).	
Register Address: 122H, 290	IA32_TSX_CTRL	
IA32_TSX_CTRL (R/W)		Thread
See Table 2-2.		
Register Address: 140H, 320	MSR_FEATURE_ENABLES	
Miscellaneous enables for thread-specific features. (R/W)		Thread

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
0	AESNI_LOCK Once this bit is set, writes to this register will not be allowed.	
63:1	Reserved.	
Register Address: 1E0H, 480	IA32_LER_INFO	
IA32_LER_INFO (R/W) Last Event Record Destination IP Register. See Table 2-2.		Thread
Register Address: 1F9H, 505	IA32_CPU_DCA_CAP	
IA32_CPU_DCA_CAP (R/O) See Table 2-2.		Thread
Register Address: 2A1H, 673	MSR_PRMRR_BASE_1	
MSR_PRMRR_BASE_1 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register.		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
19:4	Reserved.	
51:20	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2A2H, 674	MSR_PRMRR_BASE_2	
MSR_PRMRR_BASE_2 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A3H, 675	MSR_PRMRR_BASE_3	
MSR_PRMRR_BASE_3 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A4H, 676	MSR_PRMRR_BASE_4	
MSR_PRMRR_BASE_4 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A5H, 677	MSR_PRMRR_BASE_5	
MSR_PRMRR_BASE_5 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A6H, 678	MSR_PRMRR_BASE_6	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_PRMRR_BASE_6 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A7H, 679	MSR_PRMRR_BASE_7	
MSR_PRMRR_BASE_7 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2B8H, 696	MSR_COPY_SBFT_HASHES	
MSR_COPY_SBFT_HASHES (W/O)		Module
63:0	SBFT_PROGRAM_SOURCE_ADDR EDX:EAX contains the linear address base of the SBFT Binary loaded into memory.	
Register Address: 2B9H, 697	MSR_SBFT_HASHES_STATUS	
MSR_COPY_SBFT_HASHES (R/O)		Core
15:0	CHUNK_SIZE EAX[15:0] - Chunk size of the test in KB.	
31:16	TOTAL_NUM_CHUNKS EAX[31:16] - Total number of chunks.	
39:32	ERROR_CODE - EDX[7:0] The error code refers to the LP that runs WRMSR(2B8H). 0x0: Reserved. 0x1: Attempt to copy SBFT-hashes when copy already in progress. 0x2: Secure Memory not set up correctly. 0x3: Scan-Image Header Image_info.ProgramID does not match MSR_INTEGRITY_CAPABILITIES[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. 0x4: Reserved. 0x5: Integrity check failed. 0x6: WRMSR(0x2B8) (ACTIVATE_SBAF) Reinstall of SBFT test image attempted when current SBFT test image is in use by other LPs. 0x7: Aborted due to #PF (Page Fault). 0x8: Unable to generate a Random Value.	
48:40	NUM_CHUNKS_IN_STRIDE EDX[16:8] - Number of Chunks in stride. This is the number of chunks that are installed. 0 in this field means that the CPU does not support strides, otherwise, stride value must be >= 1.	
50:49	Reserved. EDX[18:17] - Set to all zeros.	
62:51	MAX_CORE_LIMIT EDX[30:19] - Maximum Number of Cores that can run SBFTAFSBAF simultaneously -1. 0 means 1 core at a time.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63	Valid. EDX[31] - Valid bit is set when COPY_SBFT_HASHES completed successfully.	
Register Address: 2BAH, 698	MSR_AUTHENTICATE_AND_COPY_SBFT_CHUNK	
MSR_AUTHENTICATE_AND_COPY_SBFT_CHUNK (w/O)		Core
63:0	BASE_CHUNK_TABLE_ADDR EDX:EAX[63:0] - Linear Address pointing to the CHUNK TABLE (TABLE_BASE).	
Register Address: 2BBH, 699	MSR_SBFT_CHUNKS_AUTHENTICATION_STATUS	
MSR_SBFT_CHUNKS_AUTHENTICATION_STATUS (R/O)		Core
15:0	NUM_VALID_CHUNKS EAX[15:0] - Total number of Valid (authenticated) chunks.	
31:16	NUM_CHUNKS_IN_STRIDE EAX[31:16] - Number of Chunks in Stride.	
39:32	ERROR_CODE EDX[7:0] 0x0 - No error reported. 0x1 - Attempt to authenticate a CHUNK already marked as authentic or is currently being installed by another core. 0x2 - CHUNK authentication error. HASH of chunk did not match expected value. 0x3 - Aborted due to #PF. 0x4 - Chunk Outside the current Stride. 0x5 - Interrupted.	
47:40	Reserved. EDX[15:8] - Set to all zeros.	
63:48	CURRENT_MAX_BUNDLE_INDX EDX[31:16] - Maximum Bundle Index in current stride.	
Register Address: 2BCH, 700	MSR_ACTIVATE_SBFT	
MSR_ACTIVATE_SBFT (w/O)		Core
13:0	SBFT_BUNDLE_INDEX EAX[13:0] - Indicates SBFT Bundle Index to start from.	
15:14	SBFT_PRGM_INDEX EAX[15:14] - Indicates what SBFT Program index to run.	
31:16	Reserved. Set to all zeros.	
62:32	THREAD_WAIT_DELAY EDX[30:0] - TSC-based delay to allow threads to rendezvous.	
63	Reserved. EDX[31] - Must be set to 0. #GP fault otherwise.	
Register Address: 2BDH, 701	MSR_SBFT_STATUS	
MSR_SBFT_STATUS (R/O)		Core

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
13:0	SBFT_BUNDLE_INDEX EAX[13:0] - SBFT Bundle that was executed.	
15:14	SBFT_PGM_INDEX EAX[15:14] - Indicates what SBFT Program index that was last ran. Maps to same field in WRMSR(ACTIVATE_SBFT). On a test pass this field will be 2'b00.	
31:16	Reserved. EAX[31:16] - Return all zeros.	
39:32	ERROR_CODE EDX[7:0] 0x0 - No Error. 0x1 - SBFT operation did not start. Other thread could not join. 0x2 - SBFT operation did not start. Interrupt occurred prior to SBFT coordination. 0x3 - Reserved. 0x4 - SBFT operation did not start. Non-valid SBFT BUNDLES in the SBFT_BUNDLE_INDEX. 0x5 - SBFT operation did not start. Mismatch in arguments between threads T0/T1. 0x6 - SBFT operation did not start. Core is not capable of performing SBFT currently. 0x7 - Reserved. 0x8 - SBFT operation did not start. Exceeded number of Logical Processors (LP) allowed to run SBFT-At-Field concurrently. 0x9 - SBFT operation did not start. Interrupt occurred or timer about to expire. 0xA - SBFT operation did not start. SBFT_PGM_INDEX is not valid. 0xB - SBFT operation aborted due to corrupted chunk. 0xC - SBFT operation did not start. TAP Data error. 0xD - SBFT operation did not start. SBFT program is not valid. All other error codes are reserved.	
60:40	Reserved. EDX[28:8] - Return all zeros.	
61	TEST_FAIL EDX[29:29] - Architectural Signature failed. Last thread executed HLT and completed SBFT and EBX != 0xACED.	
63:62	SBFT_STATUS EDX[31:30] - SBFT status (result of running SBAF). 00 - PASS. 10 - INTERRUPTED. 01 - FAILED SIGNATURE CHECK. 11 - FAILED.	
Register Address: 2BEH, 702	MSR_SBFT_MODULE_ID	
MSR_SBFT_MODULE_ID (R/O)		Module
31:0	SBFT-AT-FIELD_REVID EAX[31:0] - Maps to Revision field in external header (offset 4).	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
40:32	CURRENT_STRIDE_INDEX EDX[8:0] - Stride Index.	
63:41	Reserved. EDX[31:9] - Return all zeros.	
Register Address: 2BFH, 703	MSR_SBFTAF_LAST_WP	
MSR_SBFTAF_LAST_WP (R/O)		Module
31:0	LAST_WP EAX[31:0] - Provides information about the core when the last WRMSR(ACTIVATE_SBFT) was executed. Available only if enumerated in INTEGRITY_CAPABILITIES[10:9].	
39:32	Reserved.	
63:40	Reserved. EDX[31:8] - Return all zeros.	
Register Address: 2C2H, 706	MSR_COPY_SCAN_HASHES	
MSR_COPY_SCAN_HASHES (W/O)		Module
63:0	SCAN_HASH-ADDR EDX:EAX contains the linear address of the SCAN Test HASH Binary loaded into memory	
Register Address: 2C3H, 707	MSR_SCAN_HASHES_STATUS	
MSR_SCAN_HASHES_STATUS (R/O)		Core
15:0	CHUNK_SIZE EAX[15:0] - Chunk size of the test in KB.	
31:16	TOTAL_NUM_CHUNKS EAX[31:16] - Total number of chunks.	
39:32	ERROR_CODE EDX[7:0] - The error code refers to the LP that runs WRMSR(2C2H). ▪ 0x0 - Reserved. ▪ 0x1 - Attempt to copy scan-hashes when copy already in progress. ▪ 0x2 - Secure Memory not set up correctly. ▪ 0x3 - Scan-Image Header Image_info.ProgramID does not match MSR_INTEGRITY_CAPABILITIES[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. ▪ 0x4 - Reserved. ▪ 0x5 - Integrity check failed. ▪ 0x6 - WRMSR(0x2C6) Re-install of scan test image attempted when current scan test image is in use by other LPs. ▪ 0x7 - Aborted due to #PF (Page Fault). ▪ 0x8 - Unable to generate a Random Value.	
48:40	NUM_CHUNKS_IN_STRIDE EDX[16:8] - Number of Chunks in stride. This is the number of chunks that are installed. 0 in this field means that the CPU does not support strides, otherwise, the stride value must be >= 1.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
50:49	Reserved. EDX[18:17] - Set to all zeros.	
62:51	NAME EDX[30:19] - Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.	
63	VALID EDX[31] - Valid bit is set when COPY_SCAN_HASHES completed.	
Register Address: 2C4H, 708	MSR_AUTHENTICATE_AND_COPY_CHUNK	
MSR_AUTHENTICATE_AND_COPY_CHUNK (R/O)		Core
63:0	BASE_CHUNK_TABLE_ADDR EDX:EAX[63:0] - Linear Address pointing to the CHUNK TABLE (TABLE_BASE).	
Register Address: 2C5H, 709	MSR_CHUNKS_AUTHENTICATION_STATUS	
MSR_CHUNKS_AUTHENTICATION_STATUS (R/O)		Core
15:0	VALID_CHUNKS EAX[15:0] - Total number of Valid (authenticated) chunks.	
31:16	NUM_CHUNKS_IN_STRIDE EAX[31:16] - Number of Chunks in Stride.	
39:32	ERROR_CODE EDX[7:0] 0x0 - No-error reported. 0x1 - Attempt to authenticate a CHUNK which is already marked as authentic or is currently being installed by another core. 0x2 - CHUNK authentication error. HASH of chunk did not match expected value. 0x3 - Aborted due to #PF (Page Fault). 0x4 - Chunk Outside the current Stride.	
63:40	Reserved. EDX[31:8] - Set to all zeros.	
Register Address: 2C6H, 710	MSR_ACTIVATE_SCAN	
MSR_ACTIVATE_SCAN (W/O)		Core
15:0	CHUNK_START_INDEX EAX[15:0] - Indicates Chunk Index from which to start.	
31:16	CHUNK_STOP_INDEX EAX[31:16] - Indicates what chunk index to stop at (inclusive).	
62:32	THREAD_WAIT_DELAY EDX[30:0] - TSC based delay to allow threads to rendezvous.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63	SIGNAL_MCE EDX[31] - If 1: On scan-error log MC in MC4_STATUS and signal MCE if machine check signaling enabled in MC4_CTL[0]. - If 0: Don't no-logging/no-signaling.	
Register Address: 2C7H, 711	MSR_SCAN_STATUS	
MSR_SCAN_STATUS (R/O)		Core
15:0	CHUNK_NUM EAX[15:0] - SCAN Chunk that was reached.	
31:16	CHUNK_STOP_INDEX EAX[31:16] - Indicates what chunk index to stop at (inclusive). - Maps to same field in WRMSR(ACTIVATE_SCAN).	
39:32	ERROR_CODE EDX[7:0] 0x0 - No Error. 0x1 - SCAN operation did not start. Other thread could not join. 0x2 - SCAN operation did not start. Interrupt occurred prior to SCAN coordination. 0x3 - SCAN operation did not start. Power Management conditions are inadequate to run SAF. 0x4 - SCAN operation did not start. Non valid chunks in the range CHUNK_STOP_INDEX : CHUNK_START_INDEX. 0x5 - SCAN operation did not start. Mismatch in arguments between threads T0/T1. 0x6 - SCAN operation did not start. Core not capable of performing SCAN currently. 0x7 - Debug Mode. Scan-At-Field results not to be trusted. 0x8 - SCAN operation did not start. Exceeded number of Logical Processors (LP) allowed to run Scan-At-Field concurrently. MAX_CORE_LIMIT exceeded. 0x9 - Interrupt occurred. Scan operation aborted prematurely, not all chunks requested have been executed. 0xB - Scan operation aborted due to corrupted chunk. 0xC - Scan operation did not start. All other error codes are reserved.	
61:40	Reserved. EDX[29:8] - Return all zeros.	
62	SCAN_CONTROL_ERROR EDX[30] - SCAN error in the Scan-At-Field controller. - Non ECC error.	
63	SCAN_SIGNATURE_ERROR EDX[31] - SCAN SIGNATURE error in the SCAN pattern fetched from main memory. - Non ECC error.	
Register Address: 2C8H, 712	MSR_SCAN_MODULE_ID	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_SCAN_MODULE_ID (R/O)		Module
31:0	SCAN-AT-FIELD_REVID EAX[31:0] - Maps to Revision field in external header (offset 4).	
40:32	CURRENT_STRIDE_INDEX EDX[8:0] - Stride Index.	
63:41	Reserved. EDX[31:9] - Return all zeros.	
Register Address: 2C9H, 713		
MSR_LAST_SAF_WP (R/O)		Module
31:0	LAST_WP EAX[31:0] - Provides information about the core when the last WRMSR(ACTIVATE_SCAN) was executed. - Available only if enumerated in INTEGRITY_CAPABILITIES[10:9].	
39:32	Reserved. EDX[7:0]	
63:40	Reserved. EDX[31:8] - Return all zeros.	
Register Address: 2D9H, 729		
MSR_INTEGRITY_CAPABILITIES (R/O)		Thread
Enumerates features supported in Functional Safety.		
0	STARTUP_SCAN_BIST When set to 1, processor supports Startup SCAN BIST.	
1	STARTUP_MEM_BIST When set to 1, processor supports Startup MEM BIST.	
2	PERIODIC_MEM_BIST When set to 1, processor supports Periodic MEM BIST.	
3	LOCKSTEP When set to 1, processor supports Lock Step Mode.	
4	PERIODIC_SCAN_BIST When set to 1, processor supports Periodic SCAN BIST.	
5	PLL_LOSS_DETECT When set to 1, processor supports PLL LOSS detection.	
6	PWR_LOSS_DETECT When set to 1, processor supports Power Loss detection.	
7	PERRINJ When set to 1, processor supports FUSA PERRINJ.	
8	SBFT_AT_FIELD When set to 1, processor supports SBFT-At-Field.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
10:9	SAF_GEN_REV 00 = REV1; 01 = REV2; 10 = REV3; 11 = REV4.	
14:11	Reserved.	
15	PRESERVE_MEMORY_NEEDED When set to 1, processor supports FUSARR_BASE/MASK MSRs.	
20:16	TID_BIT_SHIFT Number of bits to shift right on x2APICID to get a unique topology ID of all logical processors that share a scan test engine.	
21	ALL_LP_JOIN_NEEDED All logical processors that share scan test engine need to be tested together and must join using MSR_ACTIVATE_SCAN.	
23:22	Reserved.	
31:24	PATTERN_ID Processor scan pattern ID. ID of the startup and periodic scan programs supported for this part.	
63:32	Reserved.	
Register Address: 30CH, 780	IA32_FIXED_CTR3	
Fixed-Function Performance Counter 3 (R/W) See Table 2-2.		Thread
Register Address: 4D0H, 1232	IA32_MCG_EXT_CTL	
IA32_MCG_EXT_CTL (R/W) See Table 2-2.		Thread
Register Address: 4F0H, 1264	MSR_SAF_CTRL	
MSR_SAF_CTRL (W/O)		Core
0	INVALIDATE_CURRENT_STRIDE EAX[0] - Write of 1 invalidates the currently installed stride. - Clears only the VALID_CHUNKS field on a RDMSR(CHUNKS_AUTHENTICATION_STATUS).	
63:1	Reserved.	
Register Address: 4F8H, 1272	MSR_SBFT_CTRL	
MSR_SBFT_CTRL (W/O)		Module
0	INVALIDATE_CURRENT_STRIDE EAX[0] - Write of 1 invalidates the currently installed stride.	
63:1	Reserved. EDX[31:0], EAX[31:1]	
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W)		Thread

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
0	WB_MEM_STRM_LD_DISABLE Disable streaming behavior for MOVNTDQA loads to WB memory type. If set, these accesses will be treated like regular cacheable loads (Data will be cached).	
63:1	Reserved.	
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W)		Core
0	SCRUB_DIS L1 scrubbing disable.	
63:1	Reserved.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY	
MSR_MC6_RESIDENCY (R/O) Time spent in the Module C6-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).		Module
63:0	RESIDENCY Time that this module is in module-specific C6 states since last reset.	
Register Address: 6E1H, 1761	IA32_PKRS	
IA32_PKRS (R/W) Specifies the PK permissions associated with each protection domain for supervisor pages. See Table 2-2.		Thread
Register Address: 7A3H, 1955	IA32_MCU_EXT_SERVICE	
MCU Extended Service MSR (R/O) If IA32_ARCH_CAPABILITIES[22] = 1. See Table 2-2.		Module
Register Address: 7A4H, 1956	IA32_MCU_ROLLBACK_MIN_ID	
Minimal MCU Revision ID for Rollback (R/O) See Table 2-2.		Module
Register Address: 7BOH, 1968	IA32_ROLLBACK_SIGN_ID_0	
Rollback ID 0 (R/O) See Table 2-2.		Module
Register Address: 7B1H, 1969	IA32_ROLLBACK_SIGN_ID_1	
Rollback ID 1 (R/O) See Table 2-2.		Module
Register Address: 7B2H, 1970	IA32_ROLLBACK_SIGN_ID_2	
Rollback ID 2 (R/O) See Table 2-2.		Module
Register Address: 7B3H, 1971	IA32_ROLLBACK_SIGN_ID_3	
Rollback ID 3 (R/O) See Table 2-2.		Module
Register Address: 7B4H, 1972	IA32_ROLLBACK_SIGN_ID_4	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Rollback ID 4 (R/O) See Table 2-2.		Module
Register Address: 7B5H, 1973	IA32_ROLLBACK_SIGN_ID_5	
Rollback ID 5 (R/O) See Table 2-2.		Module
Register Address: 7B6H, 1974	IA32_ROLLBACK_SIGN_ID_6	
Rollback ID 6 (R/O) See Table 2-2.		Module
Register Address: 7B7H, 1975	IA32_ROLLBACK_SIGN_ID_7	
Rollback ID 7 (R/O) See Table 2-2.		Module
Register Address: 7B8H, 1976	IA32_ROLLBACK_SIGN_ID_8	
Rollback ID 8 (R/O) See Table 2-2.		Module
Register Address: 7B9H, 1977	IA32_ROLLBACK_SIGN_ID_9	
Rollback ID 9 (R/O) See Table 2-2.		Module
Register Address: 7BAH, 1978	IA32_ROLLBACK_SIGN_ID_10	
Rollback ID 10 (R/O) See Table 2-2.		Module
Register Address: 7BBH, 1979	IA32_ROLLBACK_SIGN_ID_11	
Rollback ID 11 (R/O) See Table 2-2.		Module
Register Address: 7BCH, 1980	IA32_ROLLBACK_SIGN_ID_12	
Rollback ID 12 (R/O) See Table 2-2.		Module
Register Address: 7BDH, 1981	IA32_ROLLBACK_SIGN_ID_13	
Rollback ID 13 (R/O) See Table 2-2.		Module
Register Address: 7BEH, 1982	IA32_ROLLBACK_SIGN_ID_14	
Rollback ID 14 (R/O) See Table 2-2.		Module
Register Address: 7BFH, 1983	IA32_ROLLBACK_SIGN_ID_15	
Rollback ID 15 (R/O) See Table 2-2.		Module
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
IA32_TME_CAPABILITY (R/O) See Table 2-2.		Package
Register Address: 982H, 2434	IA32_TME_ACTIVATE	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
IA32_TME_ACTIVATE (R/W) See Table 2-2.		Package
Register Address: 983H, 2435	IA32_TME_EXCLUDE_MASK	
Intel TME Exclude Mask (R/W) See Table 2-2.		Package
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
Intel TME Exclude Base (R/W) See Table 2-2.		Package
Register Address: 985H, 2437	IA32_UINTR_RR	
User Interrupt Request Register (R/W) See Table 2-2.		Thread
Register Address: 986H, 2438	IA32_UINTR_HANDLER	
User Interrupt Handler Address (R/W) See Table 2-2.		Thread
Register Address: 987H, 2439	IA32_UINTR_STACKADJUST	
User Interrupt Stack Adjustment (R/W) See Table 2-2.		Thread
Register Address: 988H, 2440	IA32_UINTR_NV	
User-Interrupt Size and Notification Vector (R/W) See Table 2-2.		Thread
Register Address: 989H, 2441	IA32_UINTR_PD	
User Interrupt PID Address (R/W) See Table 2-2.		Thread
Register Address: 98AH, 2442	IA32_UINTR_TT	
User-Interrupt Target Table (R/W) See Table 2-2.		Thread
Register Address: 990H, 2448	IA32_COPY_STATUS	
IA32_COPY_STATUS (R/O) See Table 2-2.		Thread
Register Address: 991H, 2449	IA32_IWKEYBACKUP_STATUS	
IA32_IWKEYBACKUP_STATUS (R/O) See Table 2-2.		Package
Register Address: 9FBH, 2555	IA32_TME_CLEAR_SAVED_KEY	
IA32_TME_CLEAR_SAVED_KEY (R/W) See Table 2-2.		Package
Register Address: 9FFH, 2559	MSR_CORE_MKTME_ACTIVATE	
MSR to read TME_ACTIVATE[MK_TME_KEYID_BITS] (R/O)		Core
31:0	Reserved.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
35:32	READ_MK_TME_KEYID_BITS This value will be returned on a RDMSR, but must be zero on a WRMSR.	
39:36	TDX_RESERVED_KEYID_BITS (read only) The number of key identifier bits allocated to TDX usage. This is a read-only field. #GP on a non-zero write.	
63:40	Reserved.	
Register Address: C84H, 3204	MSR_MBA_CFG	
Memory Bandwidth Allocation (MBA) Configuration (R/W)		Package
1:0	Reserved.	
2	RAMBAE Resource Aware MBA Enable.	
63:3	Reserved.	
Register Address: CA0H, 3232	MSR_RMID_SNC_CONFIG	
RMID_SNC_CONFIG (R/W)		Package
0	RMID_LOCALIZED_DISTRIBUTION_MODE_ENABLE If set, Localized RMID distribution mode is enabled. If Clear, RMID Sharing mode is enabled.	
63:1	Reserved.	
Register Address: D50H, 3408	IA32_L2_QOS_EXT_BW_THRTL_0	
Memory Bandwidth Enforcement for COS0 (R/W) See Table 2-2.		Package
Register Address: D51H, 3409	IA32_L2_QOS_EXT_BW_THRTL_1	
Memory Bandwidth Enforcement for COS1 (R/W) See Table 2-2.		Package
Register Address: D52H, 3410	IA32_L2_QOS_EXT_BW_THRTL_2	
Memory Bandwidth Enforcement for COS2 (R/W) See Table 2-2.		Package
Register Address: D53H, 3411	IA32_L2_QOS_EXT_BW_THRTL_3	
Memory Bandwidth Enforcement for COS3 (R/W) See Table 2-2.		Package
Register Address: D54H, 3412	IA32_L2_QOS_EXT_BW_THRTL_4	
Memory Bandwidth Enforcement for COS4 (R/W) See Table 2-2.		Package
Register Address: D55H, 3413	IA32_L2_QOS_EXT_BW_THRTL_5	
Memory Bandwidth Enforcement for COS5 (R/W) See Table 2-2.		Package
Register Address: D56H, 3414	IA32_L2_QOS_EXT_BW_THRTL_6	
Memory Bandwidth Enforcement for COS6 (R/W) See Table 2-2.		Package

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D57H, 3415	IA32_L2_QOS_EXT_BW_THRTL_7	
Memory Bandwidth Enforcement for COS7 (R/W) See Table 2-2.		Package
Register Address: D58H, 3416	IA32_L2_QOS_EXT_BW_THRTL_8	
Memory Bandwidth Enforcement for COS8 (R/W) See Table 2-2.		Package
Register Address: D59H, 3417	IA32_L2_QOS_EXT_BW_THRTL_9	
Memory Bandwidth Enforcement for COS9 (R/W) See Table 2-2.		Package
Register Address: D5AH, 3418	IA32_L2_QOS_EXT_BW_THRTL_10	
Memory Bandwidth Enforcement for COS10 (R/W) See Table 2-2.		Package
Register Address: D5BH, 3419	IA32_L2_QOS_EXT_BW_THRTL_11	
Memory Bandwidth Enforcement for COS11 (R/W) See Table 2-2.		Package
Register Address: D5CH, 3420	IA32_L2_QOS_EXT_BW_THRTL_12	
Memory Bandwidth Enforcement for COS12 (R/W) See Table 2-2.		Package
Register Address: D5DH, 3421	IA32_L2_QOS_EXT_BW_THRTL_13	
Memory Bandwidth Enforcement for COS13 (R/W) See Table 2-2.		Package
Register Address: D5EH, 3422	IA32_L2_QOS_EXT_BW_THRTL_14	
Memory Bandwidth Enforcement for COS14 (R/W) See Table 2-2.		Package
Register Address: D91H, 3473	IA32_COPY_LOCAL_TO_PLATFORM	
See Table 2-2.		Thread
Register Address: D92H, 3474	IA32_COPY_PLATFORM_TO_LOCAL	
See Table 2-2.		Thread
Register Address: D93H, 3475	IA32_PASID	
See Table 2-2.		Thread
Register Address: 1400H, 5120	IA32_SEAMRR_BASE	
SEAM Memory Range Register for TDx - Base Address (R/W) See Table 2-2.		Core
Register Address: 1401H, 5121	IA32_SEAMRR_MASK	
SEAM Memory Range Register for TDX (R/W) See Table 2-2.		Core
Register Address: 1A8FH, 6799	MSR_STLB_QOS_INFO	
STLB_QOS_INFO (R/O) STLB QoS MASK configuration.		Core

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
5:0	NCLOS Number of CLOS supported for STLB resource using minus-1 notation.	
15:6	Reserved.	
19:16	4K_2M_CBM Length of capacity bitmask for 4K and 2M pages using minus-1 notation.	
28:20	Reserved.	
29	STLB_FILL_TRANSLATION_MSR_SUPPORTED MSR interface to fill STLB translations supported.	
30	4K_2M_ALIAS Indicates that 4K/2M pages alias into the same structure.	
63:31	Reserved.	
Register Address: 1B01H, 6913	IA32_UARCH_MISC_CTL	
IA32_UARCH_MISC_CTL (R/W) See Table 2-2.		Thread

2.17.11 MSRs Introduced in the Intel® Xeon® 6 E-Core Processors

Table 2-57 lists additional MSRs for the Intel Xeon 6 E-core processors. Intel Xeon 6 E-core processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_AFH.

For an MSR listed in Table 2-57 that also appears in the model-specific tables of prior generations, Table 2-57 supersedes prior generation tables.

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2FH, 47	IA32_BARRIER	
BARRIER (R/O) The IA32_BARRIER MSR ensures ordered execution by acting like LFENCE, controlling the sequencing of subsequent MSR reads after prior MSR reads and instructions. See Table 2-2.		Core
Register Address: 33H, 51	MSR_MEMORY_CONTROL	
Memory Control (R/W) Disables split locks, which are locked instructions that split a cache line.		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor allows one in-progress, post-retirement UC stores at a time.	
28	UC_LOCK_DISABLE If set to 1, a UC load lock will trigger a fault. If clear to 0, UC load locks proceed normally.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will trigger an #AC fault. If clear to 0, split locks proceed normally.	
63:30	Reserved.	
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/W)		Thread
31:0	SMI_COUNT Running count of SMI events since the last reset.	
63:32	Reserved.	
Register Address: 39H, 57	MSR_SOCKET_ID	
Socket ID (R/W) Reassigns the package-specific portions of the APIC ID. This MSR is used on scalable DP and high-end MP platforms to resolve legacy-mode APIC ID conflicts.		Package
10:0	PACKAGE_ID Holds package ID. This reflects the upper bits of the APIC ID.	
63:11	Reserved.	
Register Address: 7BH, 123	IA32_MCU_ENUMERATION	
Enumeration of Architectural Features (R/O) See Table 2-2.		Package
Register Address: 7CH, 124	IA32_MCU_STATUS	
MCU Status (R/O) Communicates results from the previous patch loads. See Table 2-2.		Package
Register Address: 87H, 135	IA32_MKTME_KEYID_PARTITIONING	
MKTME KEY ID Partitioning (R/O) Enumerates the number of activated KeyIDs for Intel TME-MK and Intel TDX. See Table 2-2.		Package
Register Address: 98H, 152	MSR_SEAM_WBINVDP	
SEAM WBINVDP (R/W) Allows software to WBINVD sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 99H, 153	MSR_SEAM_WBNINVDP	
SEAM WBNINVDP (R/W) Allows software to WBNINVD sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 9AH, 154	MSR_SEAM_INTR_PENDING	
SEAM Interrupt Pending (R/O) Report out some event pending bits.		Thread
0	INTR Interrupt is pending.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	NMI NMI is pending.	
2	SMI SMI is pending.	
4:3	OTHER_EVENTS Other events pending.	
63:5	Reserved.	
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL	
SMM Monitor Control (R/W) The SMM Monitor Configuration involves SMM code specifying the MSEG location and enabling dual-monitor treatment by writing to the corresponding MSR. See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information (R/O) Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
15:8	MAX_NON_TURBO_LIM_RATIO This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	
25:16	Reserved.	
26	DCU_16K_MODE_AVAIL 0b: Indicates that the part does not support the 16K DCU mode. 1b: Indicates that the part supports 16K DCU mode.	
27	Reserved.	
28	PRG_TURBO_RATIO_EN Programmable Turbo Ratios per number of Active Cores. 0 = Programming Not Allowed. 1 = Programming Allowed.	
34:29	Reserved.	
35	BIOS_GUARD_ENABLE Indicates whether the BIOS Guard feature is enabled in the CPU.	
36	PEG2DMIDIS_EN 0 = PEG2DMIDIS is disabled. 1 = PEG2DMIDIS is enabled.	
39:37	Reserved.	
47:40	MAX_EFFICIENCY_RATIO Maximum Efficiency Ratio. This is given in units of 100 MHz.	
58:48	Reserved.	
59	SMM_SUPOVR_STATE_LOCK_ENABLE When set, indicates that the CPU supports MSR SMM_SUPOVR_STATE_LOCK and the Hardware Shield feature.	
63:60	Reserved.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/W) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Core
0	STLB_QOS When set to 1, processor supports STLB QoS.	
1	Reserved.	
2	INTEGRITY_SUPPORTED When set to 1, processor supports Functional Safety. Specific FUSA capabilities are enumerated in MSR_FUSA_CAPABILITIES.	
3	RSM_IN_CPLO_ONLY Intel System Resources Defense: When set to 1, RSM will only be allowed in CPLO and will #GP for all non-CPLO privilege levels.	
4	UC_LOCK_DISABLE When set to 1, processor supports UC load lock disable.	
5	SPLIT_LOCK_DISABLE When set to 1, processor supports #AC on split locks.	
6	SNP_FILTER_QOS When set to 1, processor supports Snoop Filter Quality of Service MSRs.	
7	UC_STORE_THROTTLING When set to 1, processor supports UC store throttling through MSR_MEMORY_CTRL[UC_STORE_THROTTLE].	
63:8	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R/O) See Table 2-2.		Core
Register Address: 140H, 320	MSR_FEATURE_ENABLES	
Miscellaneous Enables for Thread-Specific Features (R/W)		Thread
0	AESNI_LOCK Once this bit is set, writes to this register will not be allowed.	
63:1	Reserved.	
Register Address: 1BOH, 432	IA32_ENERGY_PERF_BIAS	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
IA32_ENERGY_PERF_BIAS (R/W) See Table 2-2.		Thread
Register Address: 1B1H, 433	IA32_PACKAGE_THERM_STATUS	
IA32_PACKAGE_THERM_STATUS See Table 2-2.		Package
Register Address: 1B2H, 434	IA32_PACKAGE_THERM_INTERRUPT	
IA32_PACKAGE_THERM_INTERRUPT (R/W) See Table 2-2.		Package
Register Address: 2A1H, 673	MSR_PRMRR_BASE_1	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
11:4	Reserved.	
51:12	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2A2H, 674	MSR_PRMRR_BASE_2	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
11:4	Reserved.	
51:12	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2A3H, 675	MSR_PRMRR_BASE_3	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
11:4	Reserved.	
51:12	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2C2H, 706	MSR_COPY_SCAN_HASHES	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_COPY_SCAN_HASHES (W/O)		Module
63:0	SCAN_HASH-ADDR EDX:EAX contains the linear address of the SCAN Test HASH Binary loaded into memory	
Register Address: 2C3H, 707		MSR_SCAN_HASHES_STATUS
MSR_SCAN_HASHES_STATUS (R/O)		Core
15:0	CHUNK_SIZE EAX[15:0] - Chunk size of the test in KB.	
31:16	TOTAL_NUM_CHUNKS EAX[31:16] - Total number of chunks.	
39:32	ERROR_CODE EDX[7:0] - The error code refers to the LP that runs WRMSR(2C2H). 0x0 - Reserved. 0x1 - Attempt to copy scan-hashes when copy already in progress. 0x2 - Secure Memory not set up correctly. 0x3 - Scan-Image Header Image_info.ProgramID does not match MSR_INTEGRITY_CAPABILITIES[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. 0x4 - Reserved. 0x5 - Integrity check failed. 0x6 - WRMSR(0x2C6) Re-install of scan test image attempted when current scan test image is in use by other LPs. 0x7 - Aborted due to #PF (Page Fault). 0x8 - Unable to generate a Random Value.	
48:40	NUM_CHUNKS_IN_STRIDE EDX[16:8] - Number of Chunks in stride. This is the number of chunks that are installed. 0 in this field means that the CPU does not support strides, otherwise, the stride value must be >= 1	
50:49	Reserved. EDX[18:17] - Set to all zeros.	
62:51	NAME EDX[30:19] - Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.	
63	VALID EDX[31] - Valid bit is set when COPY_SCAN_HASHES completed.	
Register Address: 2C4H, 708		MSR_AUTHENTICATE_AND_COPY_CHUNK
MSR_AUTHENTICATE_AND_COPY_CHUNK(R/O)		Core
63:0	BASE_CHUNK_TABLE_ADDR EDX:EAX[63:0] - Linear Address pointing to the CHUNK TABLE (TABLE_BASE).	
Register Address: 2C5H, 709		MSR_CHUNKS_AUTHENTICATION_STATUS

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_CHUNKS_AUTHENTICATION_STATUS (R/O)		Core
15:0	VALID_CHUNKS EAX[15:0] - Total number of Valid (authenticated) chunks.	
31:16	NUM_CHUNKS_IN_STRIDE EAX[31:16] - Number of Chunks in Stride.	
39:32	ERROR_CODE EDX[7:0] ▪ 0x0 - No-error reported. ▪ 0x1 - Attempt to authenticate a CHUNK which is already marked as authentic or is currently being installed by another core. ▪ 0x2 - CHUNK authentication error. HASH of chunk did not match expected value. ▪ 0x3 - Aborted due to #PF (Page Fault). ▪ 0x4 - Chunk Outside the current Stride.	
63:40	Reserved. EDX[31:8] - Set to all zeros.	
Register Address: 2C6H, 710	MSR_ACTIVATE_SCAN	
MSR_ACTIVATE_SCAN (W/O)		Core
15:0	CHUNK_START_INDEX EAX[15:0] - Indicates Chunk Index from which to start.	
31:16	CHUNK_STOP_INDEX EAX[31:16] - Indicates what chunk index to stop at (inclusive).	
62:32	THREAD_WAIT_DELAY EDX[30:0] - TSC based delay to allow threads to rendezvous.	
63	SIGNAL_MCE EDX[31] ▪ If 1: On scan-error log MC in MC4_STATUS and signal MCE if machine check signaling enabled in MC4_CTL[0]. ▪ If 0: Don't no-logging/no-signaling.	
Register Address: 2C7H, 711	MSR_SCAN_STATUS	
MSR_SCAN_STATUS (R/O)		Core
15:0	CHUNK_NUM EAX[15:0] - SCAN Chunk that was reached.	
31:16	CHUNK_STOP_INDEX EAX[31:16] ▪ Indicates what chunk index to stop at (inclusive). ▪ Maps to same field in WRMSR(ACTIVATE_SCAN).	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
39:32	ERROR_CODE EDX[7:0] 0x0 - No Error. 0x1 - SCAN operation did not start. Other thread could not join. 0x2 - SCAN operation did not start. Interrupt occurred prior to SCAN coordination. 0x3 - SCAN operation did not start. Power Management conditions are inadequate to run SAF. 0x4 - SCAN operation did not start. Non valid chunks in the range CHUNK_STOP_INDEX : CHUNK_START_INDEX. 0x5 - SCAN operation did not start. Mismatch in arguments between threads T0/T1. 0x6 - SCAN operation did not start. Core not capable of performing SCAN currently. 0x7 - Debug Mode. Scan-At-Field results not to be trusted. 0x8 - SCAN operation did not start. Exceeded number of Logical Processors (LP) allowed to run Scan-At-Field concurrently. MAX_CORE_LIMIT exceeded. 0x9 - Interrupt occurred. Scan operation aborted prematurely, not all chunks requested have been executed. 0xB - Scan operation aborted due to corrupted chunk. 0xC - Scan operation did not start. All other error codes are reserved.	
61:40	Reserved. EDX[29:8] - Return all zeros.	
62	SCAN_CONTROL_ERROR EDX[30] - SCAN error in the Scan-At-Field controller. - Non ECC error.	
63	SCAN_SIGNATURE_ERROR EDX[31] - SCAN SIGNATURE error in the SCAN pattern fetched from main memory. - Non ECC error.	
Register Address: 2C8H, 712	MSR_SCAN_MODULE_ID	
MSR_SCAN_MODULE_ID (R/O)		Module
31:0	SCAN-AT-FIELD_REVID EAX[31:0] - Maps to Revision field in external header (offset 4).	
40:32	CURRENT_STRIDE_INDEX EDX[8:0] - Stride Index.	
63:41	Reserved. EDX[31:9] - Return all zeros.	
Register Address: 2C9H, 713	MSR_LAST_SAF_WP	
MSR_LAST_SAF_WP (R/O)		Module

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:0	LAST_WP EAX[31:0] - Provides information about the core when the last WRMSR(ACTIVATE_SCAN) was executed. - Available only if enumerated in INTEGRITY_CAPABILITIES[10:9].	
39:32	Reserved. EDX[7:0]	
63:40	Reserved. EDX[31:8] - Return all zeros.	
Register Address: 2D6H, 726	MSR_TRIGGER_PERIODIC_MEM_BIST	
MSR_TRIGGER_PERIODIC_MEM_BIST (W/O)		Core
0	SIGNAL_MCE EAX[0] - If 1, then signal MCE on fail if machine check signaling enabled in the corresponding MCI_CTL. If 0 then don't signal machine checks.	
7:1	ARRAY_BANK EAX[7:1] - Reserved.	
15:8	TST_STEP_PARAM EAX[15:8] 0: Test All Arrays, or Test Arrays in STEPs of NUM_STEPS.	
31:16	Reserved. EAX[31:16]	
63:32	Reserved. EAX[31:0]	
Register Address: 2D7H, 727	MSR_PERIODIC_MEM_BIST_STATUS	
MSR_PERIODIC_MEM_BIST_STATUS (R/O)		Core
0	MEM_BIST_STATUS 0: PASS. 1: FAIL.	
63:1	Reserved.	
Register Address: 2D9H, 729	MSR_INTEGRITY_CAPABILITIES	
MSR_INTEGRITY_CAPABILITIES (R/O) Enumerates features supported in Functional Safety.		Thread
0	STARTUP_SCAN_BIST When set to 1, processor supports Startup SCAN BIST.	
1	STARTUP_MEM_BIST When set to 1, processor supports Startup MEM BIST.	
2	PERIODIC_MEM_BIST When set to 1, processor supports Periodic MEM BIST.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3	LOCKSTEP When set to 1, processor supports Lock Step Mode.	
4	PERIODIC_SCAN_BIST When set to 1, processor supports Periodic SCAN BIST.	
5	PLL_LOSS_DETECT When set to 1, processor supports PLL LOSS detection.	
6	PWR_LOSS_DETECT When set to 1, processor supports Power Loss detection.	
7	PERRINJ When set to 1, processor supports FUSA PERRINJ.	
8	SBFT_AT_FIELD When set to 1, processor supports SBFT-At-Field.	
10:9	SAF_GEN_REV 00 = REV1; 01 = REV2; 10 = REV3; 11 = REV4.	
14:11	Reserved.	
15	PRESERVE_MEMORY_NEEDED When set to 1, processor supports FUSARR_BASE/MASK MSRs.	
20:16	TID_BIT_SHIFT Number of bits to shift right on x2APICID to get a unique topology ID of all logical processors that share a scan test engine.	
21	ALL_LP_JOIN_NEEDED All logical processors that share scan test engine need to be tested together and must join using MSR_ACTIVATE_SCAN.	
23:22	Reserved.	
31:24	PATTERN_ID Processor scan pattern ID. ID of the startup and periodic scan programs supported for this part.	
63:32	Reserved.	
Register Address: 2DCH, 732	IA32_INTEGRITY_STATUS	
IA32_INTEGRITY_STATUS (R/O) Provides status information for integrity features. See Table 2-2.		Thread
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
MSR_PKG_C6_RESIDENCY (R/O)		Package
63:0	Package C6 Residency Counter	
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
MSR_PKG_C7_RESIDENCY (R/O)		Package
63:0	Package C7 Residency Counter	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_CORE_C3_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter Time spent in the Core C-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).	
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
MSR_CORE_C6_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter Time spent in the Core C-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).	
Register Address: 3FEH, 1022	MSR_CORE_C7_RESIDENCY	
MSR_CORE_C7_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C7 Residency Counter Time spent in the Core C-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).	
Register Address: 4F0H, 1264	MSR_SAF_CTRL	
MSR_SAF_CTRL (W/O)		Core
0	INVALIDATE_CURRENT_STRIDE EAX[0] - Write of 1 invalidates the currently installed stride. - Clears only the VALID_CHUNKS field on a RDMSR(CHUNKS_AUTHENTICATION_STATUS).	
63:1	Reserved.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY	
MSR_MC6_RESIDENCY (R/O) Time spent in the Module C6-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).		Module
63:0	RESIDENCY Time that this module is in module-specific C6 states since last reset.	
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.		Thread
Register Address: 7A3H, 1955	IA32_MCU_EXT_SERVICE	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MCU Extended Service (R/W) See Table 2-2.		Module
Register Address: 7A4H, 1956	IA32_MCU_ROLLBACK_MIN_ID	
Minimal MCU Revision ID (R/O) See Table 2-2.		Module
Register Address: 7A5H, 1957	IA32_MCU_STAGING_MBOX_ADDR	
IA32_MCU_STAGING_MBOX_ADDR (R/O) See Table 2-2.		Package
Register Address: 7B0H, 1968	IA32_ROLLBACK_SIGN_ID_0	
Rollback ID 0 (R/O) See Table 2-2.		Module
Register Address: 7B1H, 1969	IA32_ROLLBACK_SIGN_ID_1	
Rollback ID 1 (R/O) See Table 2-2.		Module
Register Address: 7B2H, 1970	IA32_ROLLBACK_SIGN_ID_2	
Rollback ID 2 (R/O) See Table 2-2.		Module
Register Address: 7B3H, 1971	IA32_ROLLBACK_SIGN_ID_3	
Rollback ID 3 (R/O) See Table 2-2.		Module
Register Address: 7B4H, 1972	IA32_ROLLBACK_SIGN_ID_4	
Rollback ID 4 (R/O) See Table 2-2.		Module
Register Address: 7B5H, 1973	IA32_ROLLBACK_SIGN_ID_5	
Rollback ID 5 (R/O) See Table 2-2.		Module
Register Address: 7B6H, 1974	IA32_ROLLBACK_SIGN_ID_6	
Rollback ID 6 (R/O) See Table 2-2.		Module
Register Address: 7B7H, 1975	IA32_ROLLBACK_SIGN_ID_7	
Rollback ID 7 (R/O) See Table 2-2.		Module
Register Address: 7B8H, 1976	IA32_ROLLBACK_SIGN_ID_8	
Rollback ID 8 (R/O) See Table 2-2.		Module
Register Address: 7B9H, 1977	IA32_ROLLBACK_SIGN_ID_9	
Rollback ID 9 (R/O) See Table 2-2.		Module
Register Address: 7BAH, 1978	IA32_ROLLBACK_SIGN_ID_10	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Rollback ID 10 (R/O) See Table 2-2.		Module
Register Address: 7BBH, 1979	IA32_ROLLBACK_SIGN_ID_11	
Rollback ID 11 (R/O) See Table 2-2.		Module
Register Address: 7BCH, 1980	IA32_ROLLBACK_SIGN_ID_12	
Rollback ID 12 (R/O) See Table 2-2.		Module
Register Address: 7BDH, 1981	IA32_ROLLBACK_SIGN_ID_13	
Rollback ID 13 (R/O) See Table 2-2.		Module
Register Address: 7BEH, 1982	IA32_ROLLBACK_SIGN_ID_14	
Rollback ID 14 (R/O) See Table 2-2.		Module
Register Address: 7BFH, 1983	IA32_ROLLBACK_SIGN_ID_15	
Rollback ID 15 (R/O) See Table 2-2.		Module
Register Address: 988H, 2440	IA32_UINTR_NV	
User Interrupt Size and Notification Vector (R/W) See Table 2-2.		Thread
Register Address: 9FBH, 2555	IA32_TME_CLEAR_SAVED_KEY	
IA32_TME_CLEAR_SAVED_KEY (R/W) See Table 2-2.		Package
Register Address: 9FFH, 2559	MSR_CORE_MKTME_ACTIVATE	
MSR_CORE_MKTME_ACTIVATE (R/O) MSR to read TME_ACTIVATE[MK_TME_KEYID_BITS].		Core
31:0	Reserved.	
35:32	READ_MK_TME_KEYID_BITS This value will be returned on a RDMSR, but must be zero on a WRMSR.	
39:36	TDX_RESERVED_KEYID_BITS (read only) The number of key identifier bits allocated to TDX usage. This is a read-only field. #GP on a non-zero write.	
63:40	Reserved.	
Register Address: C84H, 3204	MSR_MBA_CFG	
Memory Bandwidth Allocation (MBA) Configuration (R/W)		Package
1:0	Reserved.	
2	RAMBAE Resource Aware MBA Enable.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:3	Reserved.	
Register Address: CA0H, 3232	MSR_RMID_SNC_CONFIG	
MSR_RMID_SNC_CONFIG (R/W)		Package
0	RMID_LOCALIZED_DISTRIBUTION_MODE_ENABLE If set, Localized RMID distribution mode is enabled. If Clear, RMID Sharing mode is enabled.	
63:1	Reserved.	
Register Address: D50H, 3408	IA32_L2_QOS_EXT_BW_THRTL_0	
Memory Bandwidth Enforcement for COS0 (R/W) See Table 2-2.		Package
Register Address: D51H, 3409	IA32_L2_QOS_EXT_BW_THRTL_1	
Memory Bandwidth Enforcement for COS1 (R/W) See Table 2-2.		Package
Register Address: D52H, 3410	IA32_L2_QOS_EXT_BW_THRTL_2	
Memory Bandwidth Enforcement for COS2 (R/W) See Table 2-2.		Package
Register Address: D53H, 3411	IA32_L2_QOS_EXT_BW_THRTL_3	
Memory Bandwidth Enforcement for COS3 (R/W) See Table 2-2.		Package
Register Address: D54H, 3412	IA32_L2_QOS_EXT_BW_THRTL_4	
Memory Bandwidth Enforcement for COS4 (R/W) See Table 2-2.		Package
Register Address: D55H, 3413	IA32_L2_QOS_EXT_BW_THRTL_5	
Memory Bandwidth Enforcement for COS5 (R/W) See Table 2-2.		Package
Register Address: D56H, 3414	IA32_L2_QOS_EXT_BW_THRTL_6	
Memory Bandwidth Enforcement for COS6 (R/W) See Table 2-2.		Package
Register Address: D57H, 3415	IA32_L2_QOS_EXT_BW_THRTL_7	
Memory Bandwidth Enforcement for COS7 (R/W) See Table 2-2.		Package
Register Address: D58H, 3416	IA32_L2_QOS_EXT_BW_THRTL_8	
Memory Bandwidth Enforcement for COS8 (R/W) See Table 2-2.		Package
Register Address: D59H, 3417	IA32_L2_QOS_EXT_BW_THRTL_9	
Memory Bandwidth Enforcement for COS9 (R/W) See Table 2-2.		Package
Register Address: D5AH, 3418	IA32_L2_QOS_EXT_BW_THRTL_10	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Memory Bandwidth Enforcement for COS10 (R/W) See Table 2-2.		Package
Register Address: D5BH, 3419	IA32_L2_QOS_EXT_BW_THRTL_11	
Memory Bandwidth Enforcement for COS11 (R/W) See Table 2-2.		Package
Register Address: D5CH, 3420	IA32_L2_QOS_EXT_BW_THRTL_12	
Memory Bandwidth Enforcement for COS12 (R/W) See Table 2-2.		Package
Register Address: D5DH, 3421	IA32_L2_QOS_EXT_BW_THRTL_13	
Memory Bandwidth Enforcement for COS13 (R/W) See Table 2-2.		Package
Register Address: D5EH, 3422	IA32_L2_QOS_EXT_BW_THRTL_14	
Memory Bandwidth Enforcement for COS14 (R/W) See Table 2-2.		Package
Register Address: E00H, 3584	IA32_QOS_CORE_BW_THRTL_0	
CBA Levels Based on COS for Bandwidth Throttling (R/W) See Table 2-2.		Thread
Register Address: E01H, 3585	IA32_QOS_CORE_BW_THRTL_1	
CBA Levels Based on COS for Bandwidth Throttling (R/W) See Table 2-2.		Thread
Register Address: 1400H, 5120	IA32_SEAMRR_BASE	
SEAM Memory Range Register for TDX - Base Address (R/W) See Table 2-2.		Core
Register Address: 1401H, 5121	IA32_SEAMRR_MASK	
SEAM Memory Range Register for TDX (R/W) See Table 2-2.		Core
Register Address: 1A8FH, 6799	MSR_STLB_QOS_INFO	
STLB_QOS_INFO (R/O) STLB QoS MASK configuration.		Core
5:0	NCLOS Number of CLOS supported for STLB resource using minus-1 notation.	
15:6	Reserved.	
19:16	4K_2M_CBM Length of capacity bitmask for 4K and 2M pages using minus-1 notation.	
28:20	Reserved.	
29	STLB_FILL_TRANSLATION_MSR_SUPPORTED MSR interface to fill STLB translations supported.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
30	4K_2M_ALIAS Indicates that 4K/2M pages alias into the same structure.	
63:31	Reserved.	

2.17.12 MSRs Introduced in the Intel® Series 2 Core™ Ultra Processor Supporting Performance Hybrid Architecture

Table 2-58 lists additional MSRs for the Intel Series 2 Core Ultra processor with a CPUID Signature DisplayFamily_DisplayModel value of 06_BDH. Table 2-59 lists the MSRs unique to the processor P-core. Table 2-60 lists the MSRs unique to the processor E-core.

For an MSR listed in Table 2-58, Table 2-59, or Table 2-60 that also appears in the model-specific tables of prior generations, Table 2-58, Table 2-59, and Table 2-60 supersede prior generation tables.

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2.		Thread
Register Address: 601H, 1537	MSR_PKG_POWER_LIMIT_4	
Package Power Limit 4 (R/W) Package-level maximum power limit (in Watts).		Package
15:0	POWER_LIMIT_4 PL4 Value in 0.125 W increments. This field is locked by PKG_POWER_LIMIT_4.LOCK. When the LOCK bit is set to 1, this field becomes Read Only. If the value is 0, PL4 limit is disabled.	
30:16	Reserved.	
31	LOCK This bit will lock the POWER_LIMIT_4 settings in this register and will also lock this setting. This means that once set to 1, the POWER_LIMIT_4 setting and this bit become Read Only until the next Warm Reset.	
63:32	Reserved.	
Register Address: 630H, 1584	MSR_PKG_C8_RESIDENCY	
MSR_PKG_C8_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
59:0	Package C8 Residency Counter Value since last reset that this package is in processor-specific C8 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 631H, 1585	MSR_PKG_C9_RESIDENCY	
MSR_PKG_C9_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C9 Residency Counter Value since last reset that this package is in processor-specific C9 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 632H, 1586	MSR_PKG_C10_RESIDENCY	
MSR_PKG_C10_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C10 Residency Counter Value since last reset that this package is in processor-specific C10 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 651H, 1617	MSR_SECONDARY_TURBO_RATIO_LIMIT_CORES	
SECONDARY_TURBO_RATIO_LIMIT_CORES (R/W) This register defines the active core ranges for each frequency point. - NUMCORE[0:7] must be populated in ascending order. - NUMCORE[i+1] must be greater than NUMCORE[i]. - Entries with NUMCORE[i] == 0 will be ignored. - The last valid entry must have NUMCORE >= the number of cores in the SKU. If any of the rules above are broken, we will silently reject the configuration.		Package
7:0	CORE_COUNT_0 Defines the active core ranges for each frequency point.	
15:8	CORE_COUNT_1 Defines the active core ranges for each frequency point.	
23:16	CORE_COUNT_2 Defines the active core ranges for each frequency point.	
31:24	CORE_COUNT_3 Defines the active core ranges for each frequency point.	
39:32	CORE_COUNT_4 Defines the active core ranges for each frequency point.	
47:40	CORE_COUNT_5 Defines the active core ranges for each frequency point.	
55:48	CORE_COUNT_6 Defines the active core ranges for each frequency point.	

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:56	CORE_COUNT_7 Defines the active core ranges for each frequency point.	
Register Address: 658H, 1624	MSR_WEIGHTED_CORE_CO	
Core-Count Weighted C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.	
Register Address: 659H, 1625	MSR_ANY_CORE_CO	
Any Core C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.	
Register Address: 65AH, 1626	MSR_ANY_GFXE_CO	
Any Graphics Engine C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is one if any processor graphic device's compute engines are in C0.	
Register Address: 65BH, 1627	MSR_CORE_GFXE_OVERLAP_CO	
Core and Graphics Engine Overlapped C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is one if at least one compute engine of the processor graphics is in C0 and at least one processor core in the package is also in C0.	
Register Address: C88H, 3208	IA32_RESOURCE_PRIORITY	
Thread scope Resource Priority Enable (R/W) See Table 2-2.		Thread
Register Address: C89H, 3209	IA32_RESOURCE_PRIORITY_PKG	
IA32_RESOURCE_PRIORITY_PKG (R/W) See Table 2-2.		Package
Register Address: 1900H, 6400	IA32_PMC_GPO_CTR	
Full Width Writable General Performance Counter 0 (R/W) See Table 2-2.		Thread
Register Address: 1901H, 6401	IA32_PMC_GPO_CFG_A	
IA32_PMC_GPO_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 0. See Table 2-2.		Thread

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1904H, 6404	IA32_PMC_GP1_CTR	
Full Width Writable General Performance Counter 1 (R/W) See Table 2-2.		Thread
Register Address: 1905H, 6405	IA32_PMC_GP1_CFG_A	
IA32_PMC_GP1_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 1. See Table 2-2.		Thread
Register Address: 1908H, 6408	IA32_PMC_GP2_CTR	
Full Width Writable General Performance Counter 2 (R/W) See Table 2-2.		Thread
Register Address: 1909H, 6409	IA32_PMC_GP2_CFG_A	
IA32_PMC_GP2_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 2. See Table 2-2.		Thread
Register Address: 190CH, 6412	IA32_PMC_GP3_CTR	
Full Width Writable General Performance Counter 3 (R/W) See Table 2-2.		Thread
Register Address: 190DH, 6413	IA32_PMC_GP3_CFG_A	
IA32_PMC_GP3_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 3. See Table 2-2.		Thread
Register Address: 1910H, 6416	IA32_PMC_GP4_CTR	
Full Width Writable General Performance Counter 4 (R/W) See Table 2-2.		Thread
Register Address: 1911H, 6417	IA32_PMC_GP4_CFG_A	
IA32_PMC_GP4_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 4. See Table 2-2.		Thread
Register Address: 1914H, 6420	IA32_PMC_GP5_CTR	
Full Width Writable General Performance Counter 5 (R/W) See Table 2-2.		Thread
Register Address: 1915H, 6421	IA32_PMC_GP5_CFG_A	
IA32_PMC_GP5_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 5. See Table 2-2.		Thread
Register Address: 1918H, 6424	IA32_PMC_GP6_CTR	
Full Width Writable General Performance Counter 6 (R/W) See Table 2-2.		Thread
Register Address: 1919H, 6425	IA32_PMC_GP6_CFG_A	

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
IA32_PMC_GP6_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 6. See Table 2-2.		Thread
Register Address: 191CH, 6428	IA32_PMC_GP7_CTR	
Full Width Writable General Performance Counter 7 (R/W) See Table 2-2.		Thread
Register Address: 191DH, 6429	IA32_PMC_GP7_CFG_A	
IA32_PMC_GP7_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 7. See Table 2-2.		Thread
Register Address: 1980H, 6528	IA32_PMC_FX0_CTR	
IA32_PMC_FX0_CTR (R/W) Fixed-Function Performance Counter 0 - Instructions Retired. See Table 2-2.		Thread
Register Address: 1984H, 6532	IA32_PMC_FX1_CTR	
IA32_PMC_FX1_CTR (R/W) Fixed-Function Performance Counter 1 - Unhalted core clock cycles. See Table 2-2.		Thread
Register Address: 1988H, 6536	IA32_PMC_FX2_CTR	
IA32_PMC_FX2_CTR (R/W) Fixed-Function Performance Counter 2 - Unhalted core reference cycles. See Table 2-2.		Thread

The MSRs listed in Table 2-59 are unique to the Intel Series 2 Core Ultra processor P-core. These MSRs are not supported on the processor E-core.

Table 2-59. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C9H, 201	IA32_PMC8	
General Performance Counter 8 (R/W) See Table 2-2.		Thread
Register Address: CAH, 202	IA32_PMC9	
General Performance Counter 9 (R/W) See Table 2-2.		Thread
Register Address: 18EH, 398	IA32_PERFEVTSEL8	
Performance Event Select Register 8 (R/W) See Table 2-2.		Thread
Register Address: 18FH, 399	IA32_PERFEVTSEL9	
Performance Event Select Register 9 (R/W) See Table 2-2.		Thread
Register Address: 30CH, 780	IA32_FIXED_CTR3	

Table 2-59. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor P-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Fixed-Function Performance Counter 3 (R/W) See Table 2-2.		Thread
Register Address: 329H, 809	MSR_PERF_METRICS	
MSR_PERF_METRICS (R/W) This register provides built-in support for Top-down Micro-architecture Analysis (TMA) metrics. It exposes the four TMA Level 1 metrics where the lower 32 bits are divided into four 8 bit fields, each of which is an integer percentage of the total TOPDOWN.SLOTS (as reported by fixed-function counter 3).		Thread
7:0	RETIRING Percent of utilized by uops that eventually retire (commit).	
15:8	BAD_SPECULATION Percent of Wasted due to incorrect speculation, covering Utilized by uops that do not retire, or Recovery Bubbles (unutilized slots).	
23:16	FRONTEND_BOUND Percent of Unutilized slots where Front-end did not deliver a uop while Back-end is ready.	
31:24	BACKEND_BOUND Percent of Unutilized slots where a uop was not delivered to Back-end due to lack of Back-end resources.	
39:32	MULTI_UOPS Frontend bound.	
47:40	BRANCH_MISPREDICTS Frontend bound.	
55:48	FRONTEND_LATENCY Frontend bound.	
63:56	MEMORY_BOUND Frontend bound.	
Register Address: 4C9H, 1225	IA32_A_PMC8	
Full Width Writable IA32_PMC8 Alias (R/W) See Table 2-2.		Thread
Register Address: 4CAH, 1226	IA32_A_PMC9	
Full Width Writable IA32_PMC9 Alias (R/W) See Table 2-2.		Thread
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Uarch Control (R/W)		Thread
0	WB_MEM_STRM_LD_DISABLE Disable streaming behavior for MOVNTDQA loads to WB memory type. If set, these accesses will be treated like regular cacheable loads (Data will be cached).	
63:1	Reserved.	
Register Address: 540H, 1344	MSR_CORE_UARCH_CTL	
Core Uarch Control (R/W)		Core

Table 2-59. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor P-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
0	SCRUB_DIS L1 scrubbing disable.	
63:1	Reserved.	
Register Address: 1920H, 6432	IA32_PMC_GP8_CTR	
Full Width Writable General Performance Counter 8 (R/W) See Table 2-2.		Thread
Register Address: 1921H, 6433	IA32_PMC_GP8_CFG_A	
IA32_PMC_GP8_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 8. See Table 2-2.		Thread
Register Address: 1924H, 6436	IA32_PMC_GP9_CTR	
Full Width Writable General Performance Counter 9 (R/W) See Table 2-2.		Thread
Register Address: 1925H, 6437	IA32_PMC_GP9_CFG_A	
IA32_PMC_GP9_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 9. See Table 2-2.		Thread
Register Address: 198CH, 6540	IA32_PMC_FX3_CTR	
IA32_PMC_FX3_CTR (R/W) See Table 2-2.		Thread

The MSRs listed in Table 2-60 are unique to the Intel Series 2 Core Ultra processor E-core. These MSRs are not supported on the processor P-core.

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2DCH, 732	IA32_INTEGRITY_STATUS	
Status Information for Integrity Features (R/O) See Table 2-2.		Thread
Register Address: 30DH, 781	IA32_FIXED_CTR4	
Fixed-Function Performance Counter 4 - Top-down Bad Speculation (R/W) See Table 2-2.		Thread
Register Address: 30EH, 782	IA32_FIXED_CTR5	
Fixed-Function Performance Counter 5 - Top-down Frontend Bound (R/W) See Table 2-2.		Thread
Register Address: 30FH, 783	IA32_FIXED_CTR6	
Fixed-Function Performance Counter 6 - Top-down Retiring (R/W) See Table 2-2.		Thread
Register Address: D18H, 3352	IA32_L2_MASK_8	

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
L2 CAT Mask for COS8 (R/W) See Table 2-2.		Module
Register Address: D19H, 3353	IA32_L2_MASK_9	
L2 CAT Mask for COS9 (R/W) See Table 2-2.		Module
Register Address: D1AH, 3354	IA32_L2_MASK_10	
L2 CAT Mask for COS10 (R/W) See Table 2-2.		Module
Register Address: D1BH, 3355	IA32_L2_MASK_11	
L2 CAT Mask for COS11 (R/W) See Table 2-2.		Module
Register Address: D1CH, 3356	IA32_L2_MASK_12	
L2 CAT Mask for COS12 (R/W) See Table 2-2.		Module
Register Address: D1DH, 3357	IA32_L2_MASK_13	
L2 CAT Mask for COS13 (R/W) See Table 2-2.		Module
Register Address: D1EH, 3358	IA32_L2_MASK_14	
L2 CAT Mask for COS14 (R/W) See Table 2-2.		Module
Register Address: D1FH, 3359	IA32_L2_MASK_15	
L2 CAT Mask for COS15 (R/W) See Table 2-2.		Module
Register Address: 1878H, 6264	MSR_WORK_CONSERVING_CLOS	
Work Conserving CLOS (R/W)		Module
0	WC_VALID WC Valid Bit that indicates WC MSR has been setup. This bit must be set for the WC algorithm to be enabled.	
7:1	Reserved.	
11:8	CLOS_START_PRI1 Starting CLOS range for priority 1.	
15:12	CLOS_END_PRI1 Ending CLOS range for priority 1.	
19:16	CLOS_START_PRI2 Starting CLOS range for priority 2.	
23:20	CLOS_END_PRI2 Ending CLOS range for priority 2.	
27:24	CLOS_START_PRI3 Starting CLOS range for priority 3.	

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:28	CLOS_END_PRI3 Ending CLOS range for priority 3.	
63:32	Reserved.	
Register Address: 1903H, 6403	IA32_PMC_GPO_CFG_C	
IA32_PMC_GPO_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1907H, 6407	IA32_PMC_GP1_CFG_C	
IA32_PMC_GP1_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 190AH, 6410	IA32_PMC_GP2_CFG_B	
IA32_PMC_GP2_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 190BH, 6411	IA32_PMC_GP2_CFG_C	
IA32_PMC_GP2_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 190EH, 6414	IA32_PMC_GP3_CFG_B	
IA32_PMC_GP3_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 190FH, 6415	IA32_PMC_GP3_CFG_C	
IA32_PMC_GP3_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1912H, 6418	IA32_PMC_GP4_CFG_B	
IA32_PMC_GP4_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 1913H, 6419	IA32_PMC_GP4_CFG_C	
IA32_PMC_GP4_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1916H, 6422	IA32_PMC_GP5_CFG_B	
IA32_PMC_GP5_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 1917H, 6423	IA32_PMC_GP5_CFG_C	
IA32_PMC_GP5_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 191AH, 6426	IA32_PMC_GP6_CFG_B	
IA32_PMC_GP6_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 191BH, 6427	IA32_PMC_GP6_CFG_C	
IA32_PMC_GP6_CFG_C (R/W) See Table 2-2.		Thread

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 191EH, 6430	IA32_PMC_GP7_CFG_B	
IA32_PMC_GP7_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 191FH, 6431	IA32_PMC_GP7_CFG_C	
IA32_PMC_GP7_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1982H, 6530	IA32_PMC_FX0_CFG_B	
Fixed-Function Counter Reload Configuration Register (R/W) See Table 2-2.		Thread
Register Address: 1983H, 6531	IA32_PMC_FX0_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 0 (R/W) See Table 2-2.		Thread
Register Address: 1986H, 6534	IA32_PMC_FX1_CFG_B	
Fixed-Function Counter Reload Configuration Register (R/W) See Table 2-2.		Thread
Register Address: 1987H, 6535	IA32_PMC_FX1_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 1 (R/W) See Table 2-2.		Thread
Register Address: 198BH, 6539	IA32_PMC_FX2_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 2 (R/W) See Table 2-2.		Thread
Register Address: 1990H, 6544	IA32_PMC_FX4_CTR	
Fixed-Function Performance Counter 4 - Top-down Bad Speculation (R/W) See Table 2-2.		Thread
Register Address: 1993H, 6547	IA32_PMC_FX4_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 4 (R/W) See Table 2-2.		Thread
Register Address: 1994H, 6548	IA32_PMC_FX5_CTR	
Fixed-Function Performance Counter 5 - Top-down Frontend Bound (R/W) See Table 2-2.		Thread
Register Address: 1997H, 6551	IA32_PMC_FX5_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 5 (R/W) See Table 2-2.		Thread
Register Address: 1998H, 6552	IA32_PMC_FX6_CTR	
Fixed-Function Performance Counter 5 - Top-down Bad Retiring (R/W) See Table 2-2.		Thread
Register Address: 199BH, 6555	IA32_PMC_FX6_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 6 (R/W) See Table 2-2.		Thread

2.18 MSRS IN THE INTEL® XEON PHI™ PROCESSOR 3200/5200/7200 SERIES AND THE INTEL® XEON PHI™ PROCESSOR 7215/7285/7295 SERIES

The Intel® Xeon Phi™ processor 3200, 5200, 7200 series, with a CPUID Signature DisplayFamily_DisplayModel value of 06_57H, supports the MSR interfaces listed in Table 2-61. These processors are based on the Knights Landing microarchitecture. The Intel® Xeon Phi™ processor 7215, 7285, 7295 series, with a CPUID Signature DisplayFamily_DisplayModel value of 06_85H, supports the MSR interfaces listed in Table 2-61 and Table 2-62. These processors are based on the Knights Mill microarchitecture. Some MSRs are shared between a pair of processor cores, and the scope is marked as module.

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, "Monitor/Mwait Address Range Determination." See Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, "Time-Stamp Counter," and Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O)	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Thread
0	Lock. (R/WL)	
1	Reserved.	
2	Enable VMX outside SMX operation. (R/WL)	
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.		Thread
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/WO) See Table 2-2.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W)		Package
2:0	Package C-State Limit (R/W) Specifies the lowest C-state for the package. This feature does not limit the processor core C-state. The power-on default value from bit[2:0] of this register reports the deepest package C-state the processor is capable to support when manufactured. It is recommended that BIOS always read the power-on default value reported from this bit field to determine the supported deepest C-state on the processor and leave it as default without changing it. 000b - C0/C1 (No package C-state support) 001b - C2 010b - C6 (non retention)* 011b - C6 (Retention)* 100b - Reserved 101b - Reserved 110b - Reserved 111b - No package C-state limit. All C-States supported by the processor are available. Note: C6 retention mode provides more power saving than C6 non-retention mode. Limiting the package to C6 non retention mode does prevent the MSR_PKG_C6_RESIDENCY counter (MSR 3F9h) from being incremented.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO registers at MSR_PMG_IO_CAPTURE_BASE[15:0] to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/O) When set, locks bits [15:0] of this register for further writes until the next reset occurs.	
25	Reserved.	
26	C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Reserved.	
28	C1 State Auto Undemotion Enable (R/W) When set, enables Undemotion from Demoted C1.	
29	PKG C-State Auto Demotion Enable (R/W) When set, enables Package C state demotion.	
63:30	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Capture Base (R/W)		Tile

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
15:0	LVL_2 Base Address (R/W) Microcode will compare IO-read zone to this base address to determine if an MWAIT(C2/3/4) needs to be issued instead of the IO-read. Should be programmed to the chipset Plevel_2 IO address.	
22:16	C-State Range (R/W) The IO-port block size in which IO-redirection will be executed (0-127). Should be programmed based on the number of LVLx registers existing in the chipset.	
63:23	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R/O) See Table 2-2.		Core
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, the AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 140H, 320	MISC_FEATURE_ENABLES	
MISC_FEATURE_ENABLES		Thread
0	Reserved.	
1	User Mode MONITOR and MWAIT (R/W) If set to 1, the MONITOR and MWAIT instructions do not cause invalid-opcode exceptions when executed with CPL > 0 or in virtual-8086 mode. If MWAIT is executed when CPL > 0 or in virtual-8086 mode, and if EAX indicates a C-state other than C0 or C1, the instruction operates as if EAX indicated the C-state C1.	
63:2	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread
Register Address: 17AH, 378	IA32_MCG_STATUS	
See Table 2-2.		Thread
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
31:0	Bank Support (SMM-RO) One bit per MCA bank. If the bit is set, that bank supports Enhanced MCA (Default all 0; does not support EMCA).	
55:32	Reserved.	
56	Targeted SMI (SMM-RO) Set if targeted SMI is supported.	
57	SMM_CPU_SVRSTR (SMM-RO) Set if SMM SRAM save/restore feature is supported.	
58	SMM_CODE_ACCESS_CHK (SMM-RO) Set if SMM code access check feature is supported.	
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.	
63:60	Reserved.	
Register Address: 186H, 390	IA32_PERFECTSELO	
Performance Monitoring Event Select Register (R/W) See Table 2-2.		Thread
7:0	Event Select.	
15:8	UMask.	
16	USR.	
17	OS.	
18	Edge.	
19	PC.	
20	INT.	
21	AnyThread.	
22	EN.	
23	INV.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:24	CMASK.	
63:32	Reserved.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Thread
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Package
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2.		Thread
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Module
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Module
0	Thermal Status (R/O)	
1	Thermal Status Log (R/WCO)	
2	PROTCHOT # or FORCEPR# Status (R/O)	
3	PROTCHOT # or FORCEPR# Log (R/WCO)	
4	Critical Temperature Status (R/O)	
5	Critical Temperature Status Log (R/WCO)	
6	Thermal Threshold #1 Status (R/O)	
7	Thermal Threshold #1 Log (R/WCO)	
8	Thermal Threshold #2 Status (R/O)	
9	Thermal Threshold #2 Log (R/WCO)	
10	Power Limitation Status (R/O)	
11	Power Limitation Log (R/WCO)	
15:12	Reserved.	
22:16	Digital Readout (R/O)	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O)	
31	Reading Valid (R/O)	
63:32	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		Thread
0	Fast-Strings Enable	
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W)	
6:4	Reserved.	
7	Performance Monitoring Available (R)	
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O)	
12	Processor Event Based Sampling Unavailable (R/O)	
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W)	
18	ENABLE MONITOR FSM (R/W)	
21:19	Reserved.	
22	Limit CPUID Maxval (R/W)	
23	xTPR Message Disable (R/W)	
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	
37:35	Reserved.	
38	Turbo Mode Disable (R/W)	
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R)	
29:24	Target Offset (R/W)	
63:30	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher.	Core
1	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher.	Core
63:2	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
Offcore Response Event Select Register (R/W)		Shared

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Shared
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode for Groups of Cores (R/W)		Package
0	Reserved.	
7:1	Maximum Number of Cores in Group 0 Number active processor cores which operates under the maximum ratio limit for group 0.	Package
15:8	Maximum Ratio Limit for Group 0 Maximum turbo ratio limit when the number of active cores are not more than the group 0 maximum core count.	Package
20:16	Number of Incremental Cores Added to Group 1 Group 1, which includes the specified number of additional cores plus the cores in group 0, operates under the group 1 turbo max ratio limit = "group 0 Max ratio limit" - "group ratio delta for group 1".	Package
23:21	Group Ratio Delta for Group 1 An unsigned integer specifying the ratio decrement relative to the Max ratio limit to Group 0.	Package
28:24	Number of Incremental Cores Added to Group 2 Group 2, which includes the specified number of additional cores plus all the cores in group 1, operates under the group 2 turbo max ratio limit = "group 1 Max ratio limit" - "group ratio delta for group 2".	Package
31:29	Group Ratio Delta for Group 2 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 1.	Package
36:32	Number of Incremental Cores Added to Group 3 Group 3, which includes the specified number of additional cores plus all the cores in group 2, operates under the group 3 turbo max ratio limit = "group 2 Max ratio limit" - "group ratio delta for group 3".	Package
39:37	Group Ratio Delta for Group 3 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 2.	Package
44:40	Number of Incremental Cores Added to Group 4 Group 4, which includes the specified number of additional cores plus all the cores in group 3, operates under the group 4 turbo max ratio limit = "group 3 Max ratio limit" - "group ratio delta for group 4".	Package
47:45	Group Ratio Delta for Group 4 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 3.	Package
52:48	Number of Incremental Cores Added to Group 5 Group 5, which includes the specified number of additional cores plus all the cores in group 4, operates under the group 5 turbo max ratio limit = "group 4 Max ratio limit" - "group ratio delta for group 5".	Package

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
55:53	Group Ratio Delta for Group 5 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 4.	Package
60:56	Number of Incremental Cores Added to Group 6 Group 6, which includes the specified number of additional cores plus all the cores in group 5, operates under the group 6 turbo max ratio limit = "group 5 Max ratio limit" - "group ratio delta for group 6".	Package
63:61	Group Ratio Delta for Group 6 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 5.	Package
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Thread
Register Address: 1B1H, 433	IA32_PACKAGE_THERM_STATUS	
See Table 2-2.		Package
Register Address: 1B2H, 434	IA32_PACKAGE_THERM_INTERRUPT	
See Table 2-2.		Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 20.9.2, "Filtering of Last Branch Records."		Thread
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W)		Thread
0	LBR Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	BTF Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.	
5:2	Reserved.	
6	TR Setting this bit to 1 enables branch trace messages to be sent.	
7	BTS Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.	
8	BTINT When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.	
9	BTS_OFF_OS When set, BTS or BTM is skipped if CPL = 0.	
10	BTS_OFF_USR When set, BTS or BTM is skipped if CPL > 0.	
11	FREEZE_LBRS_ON_PMI When set, the LBR stack is frozen on a PMI request.	
12	FREEZE_PERFMON_ON_PMI When set, each ENABLE bit of the global counter control MSR are frozen (address 3BFH) on a PMI request.	
13	Reserved.	
14	FREEZE_WHILE_SMM When set, freezes PerfMon and trace messages while in SMM.	
31:15	Reserved.	
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record from Linear IP (R)		Thread
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record to Linear IP (R)		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Core
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Core
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Core
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Core
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Core
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Core
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Core
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Core
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Core
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Core
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Core
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Core
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Core
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Core
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Core
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Core
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Core
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Core
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Core
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Core
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Core

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Core
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Core
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Core
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Core
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Core
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Core
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Core
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Core
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 20.4.1, "IA32_DEBUGCTL MSR."		Package
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2.		Thread
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2.		Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2.		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2.		Thread
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C3 Residency Counter (R/O)	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
63:0	Package C6 Residency Counter (R/O)	Package
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
63:0	Package C7 Residency Counter (R/O)	Package
Register Address: 3FCH, 1020	MSR_MCO_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Module
63:0	Module C0 Residency Counter (R/O)	
Register Address: 3FDH, 1021	MSR_MC6_RESIDENCY	
63:0	Module C6 Residency Counter (R/O)	Module
Register Address: 3FFH, 1023	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Core
63:0	CORE C6 Residency Counter (R/O)	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Thread
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Thread
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2.		Thread
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 17.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 17.10.1, "RAPL Interfaces."	Package

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:20	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C2 Residency Counter (R/O)	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W) See Section 17.10.3, "Package RAPL Domain."		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 17.10.5, "DRAM RAPL Domain."		Package
Register Address: 638H, 1592	MSR_PPO_POWER_LIMIT	
PPO RAPL Power Limit Control (R/W) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 17.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O) See Table 2-25.		Package

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
ConfigTDP Level 1 ratio and power level (R/O) See Table 2-25.		Package
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 ratio and power level (R/O) See Table 2-25.		Package
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W) See Table 2-25.		Package
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W) See Table 2-25.		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0)	
1	Thermal Status (R0)	
5:2	Reserved.	
6	VR Therm Alert Status (R0)	
7	Reserved.	
8	Electrical Design Point Status (R0)	
63:9	Reserved.	
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.		Core
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID Register (R/O)		Thread
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version Register (R/O)		Thread
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority Register (R/W)		Thread
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority Register (R/O)		Thread
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI Register (W/O)		Thread
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination Register (R/O)		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector Register (R/W)		Thread
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service Register Bits [31:0] (R/O)		Thread
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service Register Bits [63:32] (R/O)		Thread
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service Register Bits [95:64] (R/O)		Thread
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service Register Bits [127:96] (R/O)		Thread
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service Register Bits [159:128] (R/O)		Thread
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service Register Bits [191:160] (R/O)		Thread
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service Register Bits [223:192] (R/O)		Thread
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service Register Bits [255:224] (R/O)		Thread
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits [31:0] (R/O)		Thread
Register Address: 819H, 2073	IA32_X2APIC_TMR1	
x2APIC Trigger Mode Register Bits [63:32] (R/O)		Thread
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode Register Bits [95:64] (R/O)		Thread
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode Register Bits [127:96] (R/O)		Thread
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode Register Bits [159:128] (R/O)		Thread
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode Register Bits [191:160] (R/O)		Thread
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode Register Bits [223:192] (R/O)		Thread
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode Register Bits [255:224] (R/O)		Thread
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request Register Bits [31:0] (R/O)		Thread
Register Address: 821H, 2081	IA32_X2APIC_IRR1	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
x2APIC Interrupt Request Register Bits [63:32] (R/O)		Thread
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request Register Bits [95:64] (R/O)		Thread
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request Register Bits [127:96] (R/O)		Thread
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request Register Bits [159:128] (R/O)		Thread
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request Register Bits [191:160] (R/O)		Thread
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request Register Bits [223:192] (R/O)		Thread
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request Register Bits [255:224] (R/O)		Thread
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status Register (R/W)		Thread
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt Register (R/W)		Thread
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		Thread
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		Thread
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt Register (R/W)		Thread
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor Register (R/W)		Thread
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		Thread
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 Register (R/W)		Thread
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error Register (R/W)		Thread
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count Register (R/W)		Thread
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count Register (R/O)		Thread
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration Register (R/W)		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI Register (W/O)		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2		Thread

Table 2-62 lists model-specific registers that are supported by the Intel® Xeon Phi™ processor 7215, 7285, 7295 series based on the Knights Mill microarchitecture.

Table 2-62. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL	
SMM Monitor Configuration (R/W) This MSR is readable only if VMX is enabled, and writeable only if VMX is enabled and in SMM mode, and is used to configure the VMX MSEG base address. See Table 2-2.		Core
Register Address: 480H, 1152	IA32_VMX_BASIC	

Table 2-62. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2.		Core
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS	
Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2.		Core
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)		Core
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS	
Capability Reporting Register of VM-exit Controls (R/O) See Table 2-2.		Core
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-entry Controls (R/O) See Table 2-2.		Core
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2.		Core
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2.		Core
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2.		Core
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2.		Core
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2.		Core
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2.		Core
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Table 2-2.		Core
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.		Core
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTLS	

Table 2-62. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTLS	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2.		Core
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2.		Core
Register Address: 491H, 1169	IA32_VMX_FMFUNC	
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Core

2.19 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table 2-63 lists MSRs (architectural and model-specific) that are defined across processor generations based on Intel NetBurst microarchitecture. The processor can be identified by its CPUID signatures of DisplayFamily encoding of 0FH, see Table 2-1.

- MSRs with an "IA32_" prefix are designated as "architectural." This means that the functions of these MSRs and their addresses remain the same for succeeding families of IA-32 processors.
- MSRs with an "MSR_" prefix are model specific with respect to address functionalities. The column "Model Availability" lists the model encoding value(s) within the Pentium 4 and Intel Xeon processor family at the specified register address. The model encoding value of a processor can be queried using CPUID. See "CPUID—CPU Identification" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/Unique ¹
Register Address: 0H, 0	IA32_P5_MC_ADDR		
See Section 2.23, "MSRs in Pentium Processors."		0, 1, 2, 3, 4, 6	Shared
Register Address: 1H, 1	IA32_P5_MC_TYPE		
See Section 2.23, "MSRs in Pentium Processors."		0, 1, 2, 3, 4, 6	Shared
Register Address: 6H, 6	IA32_MONITOR_FILTER_LINE_SIZE		
See Section 11.10.5, "Monitor/Mwait Address Range Determination."		3, 4, 6	Shared
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER		
Time Stamp Counter See Table 2-2.		0, 1, 2, 3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
On earlier processors, only the lower 32 bits are writable. On any write to the lower 32 bits, the upper 32 bits are cleared. For processor family 0FH, models 3 and 4: all 64 bits are writable.			
Register Address: 17H, 23	IA32_PLATFORM_ID		
Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.		0, 1, 2, 3, 4, 6	Shared
Register Address: 1BH, 27	IA32_APIC_BASE		
APIC Location and Status (R/W) See Table 2-2. See Section 13.4.4, "Local APIC Status and Location."		0, 1, 2, 3, 4, 6	Unique
Register Address: 2AH, 42	MSR_EBC_HARD_POWERON		
Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.		0, 1, 2, 3, 4, 6	Shared
0	Output Tri-state Enabled (R) Indicates whether tri-state output is enabled (1) or disabled (0) as set by the strapping of SMI#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
1	Execute BIST (R) Indicates whether the execution of the BIST is enabled (1) or disabled (0) as set by the strapping of INIT#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
2	In Order Queue Depth (R) Indicates whether the in order queue depth for the system bus is 1 (1) or up to 12 (0) as set by the strapping of A7#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
3	MCERR# Observation Disabled (R) Indicates whether MCERR# observation is enabled (0) or disabled (1) as determined by the strapping of A9#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
4	BINIT# Observation Enabled (R) Indicates whether BINIT# observation is enabled (0) or disabled (1) as determined by the strapping of A10#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
6:5	APIC Cluster ID (R) Contains the logical APIC cluster ID value as set by the strapping of A12# and A11#. The logical cluster ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
7	Bus Park Disable (R) Indicates whether bus park is enabled (0) or disabled (1) as set by the strapping of A15#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
11:8	Reserved.		
13:12	Agent ID (R) Contains the logical agent ID value as set by the strapping of BR[3:0]. The logical ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.		
63:14	Reserved.		
Register Address: 2BH, 43	MSR_EBC_SOFT_POWERON		
Processor Soft Power-On Configuration (R/W) Enables and disables processor features.		0, 1, 2, 3, 4, 6	Shared
0	RCNT/SCNT On Request Encoding Enable (R/W) Controls the driving of RCNT/SCNT on the request encoding. Set to enable (1); clear to disabled (0, default).		
1	Data Error Checking Disable (R/W) Set to disable system data bus parity checking; clear to enable parity checking.		
2	Response Error Checking Disable (R/W) Set to disable (default); clear to enable.		
3	Address/Request Error Checking Disable (R/W) Set to disable (default); clear to enable.		
4	Initiator MCERR# Disable (R/W) Set to disable MCERR# driving for initiator bus requests (default); clear to enable.		
5	Internal MCERR# Disable (R/W) Set to disable MCERR# driving for initiator internal errors (default); clear to enable.		
6	BINIT# Driver Disable (R/W) Set to disable BINIT# driver (default); clear to enable driver.		
63:7	Reserved.		
Register Address: 2CH, 44	MSR_EBC_FREQUENCY_ID		
Processor Frequency Configuration The bit field layout of this MSR varies according to the MODEL value in the CPUID version information. The following bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding equal or greater than 2. (R) The field Indicates the current processor frequency configuration.		2,3, 4, 6	Shared
15:0	Reserved.		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
18:16	Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz (Model 2) 000B 266 MHz (Model 3 or 4) 001B 133 MHz 010B 200 MHz 011B 166 MHz 100B 333 MHz (Model 6)		
	133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.		
	266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B and model encoding = 3 or 4. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B and model encoding = 6. All other values are reserved.		
23:19	Reserved.		
31:24	Core Clock Frequency to System Bus Frequency Ratio (R) The processor core clock frequency to system bus frequency ratio observed at the deassertion of the reset pin.		
63:32	Reserved.		
Register Address: 2CH, 44	MSR_EBC_FREQUENCY_ID		
Processor Frequency Configuration (R) The bit field layout of this MSR varies according to the MODEL value of the CPUID version information. This bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding less than 2. Indicates current processor frequency configuration.		0, 1	Shared
20:0	Reserved.		
23:21	Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz All others values reserved.		
63:24	Reserved.		
Register Address: 3AH, 58	IA32_FEATURE_CONTROL		
Control Features in IA-32 Processor (R/W) See Table 2-2. (If CPUID.01H:ECX[5])		3, 4, 6	Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
BIOS Update Trigger Register (W) See Table 2-2.		0, 1, 2, 3, 4, 6	Shared
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID		
BIOS Update Signature ID (R/W) See Table 2-2.		0, 1, 2, 3, 4, 6	Unique
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL		
SMM Monitor Configuration (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: FEH, 254	IA32_MTRRCAP		
MTRR Information See Section 14.11.1, "MTRR Feature Identification."		0, 1, 2, 3, 4, 6	Unique
Register Address: 174H, 372	IA32_SYSENTER_CS		
CS Register Target for CPL 0 Code (R/W) See Table 2-2 and Section 6.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP		
Stack Pointer for CPL 0 Stack (R/W) See Table 2-2 and Section 6.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP		
CPL 0 Code Entry Point (R/W) See Table 2-2 and Section 6.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 179H, 377	IA32_MCG_CAP		
Machine Check Capabilities (R) See Table 2-2 and Section 18.3.1.1, "IA32_MCG_CAP MSR."		0, 1, 2, 3, 4, 6	Unique
Register Address: 17AH, 378	IA32_MCG_STATUS		
Machine Check Status (R) See Table 2-2 and Section 18.3.1.2, "IA32_MCG_STATUS MSR."		0, 1, 2, 3, 4, 6	Unique
Register Address: 17BH, 379	IA32_MCG_CTL		
Machine Check Feature Enable (R/W) See Table 2-2 and Section 18.3.1.3, "IA32_MCG_CTL MSR."			
Register Address: 180H, 384	MSR_MCG_RAX		
Machine Check EAX/RAX Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 181H, 385	MSR_MCG_RBX		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Machine Check EBX/RBX Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 182H, 386	MSR_MCG_RCX		
Machine Check ECX/RCX Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 183H, 387	MSR_MCG_RDX		
Machine Check EDX/RDX Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 184H, 388	MSR_MCG_RSI		
Machine Check ESI/RSI Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 185H, 389	MSR_MCG_RDI		
Machine Check EDI/RDI Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 186H, 390	MSR_MCG_RBP		
Machine Check EBP/RBP Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 187H, 391	MSR_MCG_RSP		
Machine Check ESP/RSP Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 188H, 392	MSR_MCG_RFLAGS		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Machine Check EFLAGS/RFLAG Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 189H, 393	MSR_MCG_RIP		
Machine Check EIP/RIP Save State See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 18AH, 394	MSR_MCG_MISC		
Machine Check Miscellaneous See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
0	DS When set, the bit indicates that a page assist or page fault occurred during DS normal operation. The processors response is to shut down. The bit is used as an aid for debugging DS handling code. It is the responsibility of the user (BIOS or operating system) to clear this bit for normal operation.		
63:1	Reserved.		
Register Address: 18BH–18FH, 395–399	MSR_MCG_RESERVED1–MSR_MCG_RESERVED5		
Reserved.			
Register Address: 190H, 400	MSR_MCG_R8		
Machine Check R8 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 191H, 401	MSR_MCG_R9		
Machine Check R9D/R9 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 192H, 402	MSR_MCG_R10		
Machine Check R10 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 193H, 403	MSR_MCG_R11		
Machine Check R11 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 194H, 404	MSR_MCG_R12		
Machine Check R12 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 195H, 405	MSR_MCG_R13		
Machine Check R13 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 196H, 406	MSR_MCG_R14		
Machine Check R14 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 197H, 407	MSR_MCG_R15		
Machine Check R15 See Section 18.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 198H, 408	IA32_PERF_STATUS		
See Table 2-2. See Section 17.1, "Enhanced Intel Speedstep® Technology."		3, 4, 6	Unique
Register Address: 199H, 409	IA32_PERF_CTL		
See Table 2-2. See Section 17.1, "Enhanced Intel Speedstep® Technology."		3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 19AH, 410	IA32_CLOCK_MODULATION		
Thermal Monitor Control (R/W) See Table 2-2 and Section 17.8.3, "Software Controlled Clock Modulation."		0, 1, 2, 3, 4, 6	Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT		
Thermal Interrupt Control (R/W) See Section 17.8.2, "Thermal Monitor," and Table 2-2.		0, 1, 2, 3, 4, 6	Unique
Register Address: 19CH, 412	IA32_THERM_STATUS		
Thermal Monitor Status (R/W) See Section 17.8.2, "Thermal Monitor," and Table 2-2.		0, 1, 2, 3, 4, 6	Shared
Register Address: 19DH, 413	MSR_THERM2_CTL		
Thermal Monitor 2 Control			
For Family F, Model 3 processors: When read, specifies the value of the target TM2 transition last written. When set, it sets the next target value for TM2 transition.		3	Shared
For Family F, Model 4 and Model 6 processors: When read, specifies the value of the target TM2 transition last written. Writes may cause #GP exceptions.		4, 6	Shared
Register Address: 1A0H, 416	IA32_MISC_ENABLE		
Enable Miscellaneous Processor Features (R/W)		0, 1, 2, 3, 4, 6	Shared
0	Fast-Strings Enable. See Table 2-2.		
1	Reserved.		
2	x87 FPU Fopcode Compatibility Mode Enable		
3	Thermal Monitor 1 Enable See Section 17.8.2, "Thermal Monitor," and Table 2-2.		
4	Split-Lock Disable When set, the bit causes an #AC exception to be issued instead of a split-lock cycle. Operating systems that set this bit must align system structures to avoid split-lock scenarios. When the bit is clear (default), normal split-locks are issued to the bus. This debug feature is specific to the Pentium 4 processor.		
5	Reserved.		
6	Third-Level Cache Disable (R/W) When set, the third-level cache is disabled; when clear (default) the third-level cache is enabled. This flag is reserved for processors that do not have a third-level cache. Note that the bit controls only the third-level cache; and only if overall caching is enabled through the CD flag of control register CR0, the page-level cache controls, and/or the MTRRs. See Section 14.5.4, "Disabling and Enabling the L3 Cache."		
7	Performance Monitoring Available (R) See Table 2-2.		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
8	Suppress Lock Enable When set, assertion of LOCK on the bus is suppressed during a Split Lock access. When clear (default), LOCK is not suppressed.		
9	Prefetch Queue Disable When set, disables the prefetch queue. When clear (default), enables the prefetch queue.		
10	FERR# Interrupt Reporting Enable (R/W) When set, interrupt reporting through the FERR# pin is enabled; when clear, this interrupt reporting function is disabled. When this flag is set and the processor is in the stop-clock state (STPCLK# is asserted), asserting the FERR# pin signals to the processor that an interrupt (such as, INIT#, BINIT#, INTR, NMI, SMI#, or RESET#) is pending and that the processor should return to normal operation to handle the interrupt. This flag does not affect the normal operation of the FERR# pin (to indicate an unmasked floating-point error) when the STPCLK# pin is not asserted.		
11	Branch Trace Storage Unavailable (BTS_UNAVAILABLE) (R) See Table 2-2. When set, the processor does not support branch trace storage (BTS); when clear, BTS is supported.		
12	PEBS_UNAVAILABLE: Processor Event Based Sampling Unavailable (R) See Table 2-2. When set, the processor does not support processor event-based sampling (PEBS); when clear, PEBS is supported.		
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.	3	
17:14	Reserved.		
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	3, 4, 6	

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
19	Adjacent Cache Line Prefetch Disable (R/W) When set to 1, the processor fetches the cache line of the 128-byte sector containing currently required data. When set to 0, the processor fetches both cache lines in the sector. Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.		
21:20	Reserved.		
22	Limit CPUID MAXVAL (R/W) See Table 2-2. Setting this can cause unexpected behavior to software that depends on the availability of CPUID leaves greater than 3.	3, 4, 6	
23	xTPR Message Disable (R/W) See Table 2-2.		Shared
24	L1 Data Cache Context Mode (R/W) When set, the L1 data cache is placed in shared mode; when clear (default), the cache is placed in adaptive mode. This bit is only enabled for IA-32 processors that support Intel Hyper-Threading Technology. See Section 14.5.6, "L1 Data Cache Context Mode." When L1 is running in adaptive mode and CR3s are identical, data in L1 is shared across logical processors. Otherwise, L1 is not shared and cache use is competitive. If the Context ID feature flag (ECX[10]) is set to 0 after executing CPUID with EAX = 1, the ability to switch modes is not supported. BIOS must not alter the contents of IA32_MISC_ENABLE[24].		
33:25	Reserved.		
34	XD Bit Disable (R/W) See Table 2-3.		Unique
63:35	Reserved.		
Register Address: 1A1H, 417	MSR_PLATFORM_BRV		
Platform Feature Requirements (R)		3, 4, 6	Shared
17:0	Reserved.		
18	PLATFORM Requirements When set to 1, indicates the processor has specific platform requirements. The details of the platform requirements are listed in the respective data sheets of the processor.		
63:19	Reserved.		
Register Address: 1D7H, 471	MSR_LER_FROM_LIP		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 20.13.3, "Last Exception Records."		0, 1, 2, 3, 4, 6	Unique
31:0	From Linear IP Linear address of the last branch instruction.		
63:32	Reserved.		
Register Address: 1D7H, 471	MSR_LER_FROM_LIP		
63:0	From Linear IP Linear address of the last branch instruction (If IA-32e mode is active).		Unique
Register Address: 1D8H, 472	MSR_LER_TO_LIP		
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 20.13.3, "Last Exception Records."		0, 1, 2, 3, 4, 6	Unique
31:0	From Linear IP Linear address of the target of the last branch instruction.		
63:32	Reserved.		
Register Address: 1D8H, 472	MSR_LER_TO_LIP		
63:0	From Linear IP Linear address of the target of the last branch instruction (If IA-32e mode is active).		Unique
Register Address: 1D9H, 473	MSR_DEBUGCTLA		
Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 20.13.1, "MSR_DEBUGCTLA MSR."		0, 1, 2, 3, 4, 6	Unique
Register Address: 1DAH, 474	MSR_LASTBRANCH_TOS		
Last Branch Record Stack TOS (R/O) Contains an index (0-3 or 0-15) that points to the top of the last branch record stack (that is, that points the index of the MSR containing the most recent branch record). See Section 20.13.2, "LBR Stack for Processors Based on Intel NetBurst® Microarchitecture," and addresses 1DBH-1DEH and 680H-68FH.		0, 1, 2, 3, 4, 6	Unique
Register Address: 1DBH, 475	MSR_LASTBRANCH_0		
Last Branch Record 0 (R/O) One of four last branch record registers on the last branch record stack. It contains pointers to the source and destination instruction for one of the last four branches, exceptions, or interrupts that the processor took. MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3 at 1DBH-1DEH are available only on family 0FH, models 0H-02H. They have been replaced by the MSRs at 680H-68FH and 6C0H-6CFH. See Section 20.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."		0, 1, 2	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 1DCH, 476	MSR_LASTBRANCH_1		
Last Branch Record 1 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 1DDH, 477	MSR_LASTBRANCH_2		
Last Branch Record 2 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 1DEH, 478	MSR_LASTBRANCH_3		
Last Branch Record 3 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0		
Variable Range Base MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7		
Variable Range Mask MTRR See Section 14.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000		
Fixed Range MTRR See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 277H, 631	IA32_PAT		
Page Attribute Table See Section 14.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE		
Default Memory Types (R/W) See Table 2-2 and Section 14.11.2.1, "IA32_MTRR_DEF_TYPE MSR."		0, 1, 2, 3, 4, 6	Shared
Register Address: 300H, 768	MSR_BPU_COUNTER0		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 301H, 769	MSR_BPU_COUNTER1		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 302H, 770	MSR_BPU_COUNTER2		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 303H, 771	MSR_BPU_COUNTER3		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 304H, 772	MSR_MS_COUNTER0		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 305H, 773	MSR_MS_COUNTER1		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 306H, 774	MSR_MS_COUNTER2		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 307H, 775	MSR_MS_COUNTER3		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 308H, 776	MSR_FLAME_COUNTER0		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 309H, 777	MSR_FLAME_COUNTER1		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30AH, 778	MSR_FLAME_COUNTER2		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30BH, 779	MSR_FLAME_COUNTER3		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30CH, 780	MSR_IQ_COUNTER0		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30DH, 781	MSR_IQ_COUNTER1		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30EH, 782	MSR_IQ_COUNTER2		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30FH, 783	MSR_IQ_COUNTER3		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 310H, 784	MSR_IQ_COUNTER4		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 311H, 785	MSR_IQ_COUNTER5		
See Section 22.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 360H, 864	MSR_BPU_CCCR0		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 361H, 865	MSR_BPU_CCCR1		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 362H, 866	MSR_BPU_CCCR2		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 363H, 867	MSR_BPU_CCCR3		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 364H, 868	MSR_MS_CCCR0		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 365H, 869	MSR_MS_CCCR1		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 366H, 870	MSR_MS_CCCR2		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 367H, 871	MSR_MS_CCCR3		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 368H, 872	MSR_FLAME_CCCR0		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 369H, 873	MSR_FLAME_CCCR1		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36AH, 874	MSR_FLAME_CCCR2		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36BH, 875	MSR_FLAME_CCCR3		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36CH, 876	MSR_IQ_CCCR0		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36DH, 877	MSR_IQ_CCCR1		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36EH, 878	MSR_IQ_CCCR2		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36FH, 879	MSR_IQ_CCCR3		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 370H, 880	MSR_IQ_CCCR4		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 371H, 881	MSR_IQ_CCCR5		
See Section 22.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A0H, 928	MSR_BSU_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A1H, 929	MSR_BSU_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A2H, 930	MSR_FSB_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A3H, 931	MSR_FSB_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A4H, 932	MSR_FIRM_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A5H, 933	MSR_FIRM_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A6H, 934	MSR_FLAME_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A7H, 935	MSR_FLAME_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A8H, 936	MSR_DAC_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A9H, 937	MSR_DAC_ESCR1		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AAH, 938	MSR_MOB_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ABH, 939	MSR_MOB_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ACH, 940	MSR_PMH_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ADH, 941	MSR_PMH_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AEH, 942	MSR_SAAT_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AFH, 943	MSR_SAAT_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BOH, 944	MSR_U2L_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B1H, 945	MSR_U2L_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B2H, 946	MSR_BPU_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B3H, 947	MSR_BPU_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B4H, 948	MSR_IS_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B5H, 949	MSR_IS_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B6H, 950	MSR_ITLB_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B7H, 951	MSR_ITLB_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B8H, 952	MSR_CRU_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B9H, 953	MSR_CRU_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BAH, 954	MSR_IQ_ESCR0		
See Section 22.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.		0, 1, 2	Shared

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 3BBH, 955	MSR_IQ_ESCR1		
See Section 22.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.		0, 1, 2	Shared
Register Address: 3BCH, 956	MSR_RAT_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BDH, 957	MSR_RAT_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BEH, 958	MSR_SSU_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3COH, 960	MSR_MS_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C1H, 961	MSR_MS_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C2H, 962	MSR_TBPU_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C3H, 963	MSR_TBPU_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C4H, 964	MSR_TC_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C5H, 965	MSR_TC_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C8H, 968	MSR_IX_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C9H, 969	MSR_IX_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CAH, 970	MSR_ALF_ESCR0		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CBH, 971	MSR_ALF_ESCR1		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CCH, 972	MSR_CRU_ESCR2		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CDH, 973	MSR_CRU_ESCR3		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3E0H, 992	MSR_CRU_ESCR4		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3E1H, 993	MSR_CRU_ESCR5		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3F0H, 1008	MSR_TC_PRECISE_EVENT		
See Section 22.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)		
Processor Event Based Sampling (PEBS) (R/W) Controls the enabling of processor event sampling and replay tagging.		0, 1, 2, 3, 4, 6	Shared
12:0	See https://perfmon-events.intel.com/ .		
23:13	Reserved.		
24	UOP Tag Enables replay tagging when set.		
25	ENABLE_PEBS_MY_THR (R/W) Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 22.6.4.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is called ENABLE_PEBS in IA-32 processors that do not support Intel Hyper-Threading Technology.		
26	ENABLE_PEBS_OTH_THR (R/W) Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 22.6.4.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is reserved for IA-32 processors that do not support Intel Hyper-Threading Technology.		
63:27	Reserved.		
Register Address: 3F2H, 1010	MSR_PEBS_MATRIX_VERT		
See https://perfmon-events.intel.com/ .		0, 1, 2, 3, 4, 6	Shared
Register Address: 400H, 1024	IA32_MCO_CTL		
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 401H, 1025	IA32_MCO_STATUS		
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 402H, 1026	IA32_MCO_ADDR		
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 403H, 1027	IA32_MCO_MISC		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 18.3.2.4, "IA32_MCI_MISC MSRs." The IA32_MCO_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 404H, 1028	IA32_MC1_CTL		
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 405H, 1029	IA32_MC1_STATUS		
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 406H, 1030	IA32_MC1_ADDR		
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 407H, 1031	IA32_MC1_MISC		
See Section 18.3.2.4, "IA32_MCI_MISC MSRs." The IA32_MC1_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			Shared
Register Address: 408H, 1032	IA32_MC2_CTL		
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 409H, 1033	IA32_MC2_STATUS		
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 40AH, 1034	IA32_MC2_ADDR		
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 40BH, 1035	IA32_MC2_MISC		
See Section 18.3.2.4, "IA32_MCI_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 40CH, 1036	IA32_MC3_CTL		
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 40DH, 1037	IA32_MC3_STATUS		
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 40EH, 1038	IA32_MC3_ADDR		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 40FH, 1039	IA32_MC3_MISC		
See Section 18.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC3_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 410H, 1040	IA32_MC4_CTL		
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 411H, 1041	IA32_MC4_STATUS		
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 412H, 1042	IA32_MC4_ADDR		
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 413H, 1043	IA32_MC4_MISC		
See Section 18.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 480H, 1152	IA32_VMX_BASIC		
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		3, 4, 6	Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL		
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		3, 4, 6	Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL		
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL		
Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 485H, 1157	IA32_VMX_MISC		
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data," and Table 2-2.		3, 4, 6	Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0		
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and Table 2-2.		3, 4, 6	Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1		
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and Table 2-2.		3, 4, 6	Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0		
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and Table 2-2.		3, 4, 6	Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1		
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and Table 2-2.		3, 4, 6	Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM		
Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration," and Table 2-2.		3, 4, 6	Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLX2		
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 600H, 1536	IA32_DS_AREA		
DS Save Area (R/W) See Table 2-2 and Section 22.6.3.4, "Debug Store (DS) Mechanism."		0, 1, 2, 3, 4, 6	Unique
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP		
Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (680H-68FH). This part of the stack contains pointers to the source instruction for one of the last 16 branches, exceptions, or interrupts taken by the processor. The MSRs at 680H-68FH, 6C0H-6CfH are not available in processor releases before family 0FH, model 03H. These MSRs replace MSRs previously located at 1DBH-1DEH, which performed the same function for early releases. See Section 20.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."		3, 4, 6	Unique
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP		
Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP		
Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP		
Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP		
Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP		
Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP		
Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP		
Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP		
Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP		
Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP		
Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP		
Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP		
Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP		
Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP		
Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP		
Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (6C0H-6CFH). This part of the stack contains pointers to the destination instruction for one of the last 16 branches, exceptions, or interrupts that the processor took. See Section 20.12, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture.”		3, 4, 6	Unique
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP		
Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP		
Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP		
Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP		
Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP		
Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP		
Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP		
Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP		
Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP		
Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP		
Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP		
Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP		
Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP		
Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP		
Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: C000_0080H	IA32_EFER		
Extended Feature Enables See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0081H	IA32_STAR		
System Call Target Address (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0082H	IA32_LSTAR		
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0084H	IA32_FMASK		
System Call Flag Mask (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0100H	IA32_FS_BASE		
Map of BASE Address of FS (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0101H	IA32_GS_BASE		
Map of BASE Address of GS (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE		
Swap Target of BASE Address of GS (R/W) See Table 2-2.		3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
NOTES 1. For HT-enabled processors, there may be more than one logical processors per physical unit. If an MSR is Shared, this means that one MSR is shared between logical processors. If an MSR is unique, this means that each logical processor has its own MSR.			

2.19.1 MSRs Unique to Intel® Xeon® Processor MP with L3 Cache

The MSRs listed in Table 2-64 apply to Intel® Xeon® Processor MP with up to 8MB level three cache. These processors can be detected by enumerating the deterministic cache parameter leaf, CPUID.04H, to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 3 or 4 (see CPUID instruction for more details).

Table 2-64. MSRs Unique to 64-bit Intel® Xeon® Processor MP with Up to an 8 MB L3 Cache

Register Address: Hex	Register Name		
Register Information		Model Availability	Shared/ Unique
Register Address: 107CCH	MSR_IFSB_BUSQ0		
IFSB BUSQ Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107CDH	MSR_IFSB_BUSQ1		
IFSB BUSQ Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107CEH	MSR_IFSB_SNPQ0		
IFSB SNPQ Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107CFH	MSR_IFSB_SNPQ1		
IFSB SNPQ Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107D0H	MSR_EFSB_DRDY0		
EFSB DRDY Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107D1H	MSR_EFSB_DRDY1		
EFSB DRDY Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107D2H	MSR_IFSB_CTL6		
IFSB Latency Event Control Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107D3H	MSR_IFSB_CNTR7		
IFSB Latency Event Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared

The MSRs listed in Table 2-65 apply to Intel® Xeon® Processor 7100 series. These processors can be detected by enumerating the deterministic cache parameter, CPUID.04H, to detect the presence of the third level cache, and

with CPUID reporting family encoding 0FH, model encoding 6 (See CPUID instruction for more details.). The performance monitoring MSRs listed in Table 2-65 are shared between logical processors in the same core, but are replicated for each core.

Table 2-65. MSRs Unique to Intel® Xeon® Processor 7100 Series

Register Address: Hex	Register Name		
Register Information		Model Availability	Shared/Unique
Register Address: 107CCH	MSR_EMON_L3_CTR_CTL0		
GBUSQ Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		6	Shared
Register Address: 107CDH	MSR_EMON_L3_CTR_CTL1		
GBUSQ Event Control and Counter Register (R/W)		6	Shared
Register Address: 107CEH	MSR_EMON_L3_CTR_CTL2		
GSPNQ Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		6	Shared
Register Address: 107CFH	MSR_EMON_L3_CTR_CTL3		
GSPNQ Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D0H	MSR_EMON_L3_CTR_CTL4		
FSB Event Control and Counter Register (R/W) See Section 22.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		6	Shared
Register Address: 107D1H	MSR_EMON_L3_CTR_CTL5		
FSB Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D2H	MSR_EMON_L3_CTR_CTL6		
FSB Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D3H	MSR_EMON_L3_CTR_CTL7		
FSB Event Control and Counter Register (R/W)		6	Shared

2.20 MSRS IN INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS

Model-specific registers (MSRs) for Intel Core Solo, Intel Core Duo processors, and Dual-core Intel Xeon processor LV are listed in Table 2-66. The column "Shared/Unique" applies to Intel Core Duo processor. "Unique" means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. "Shared" means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/Unique
Register Address: 0H, 0	P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors," and Table 2-2.		Unique
Register Address: 1H, 1	P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors," and Table 2-2.		Unique

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 11.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 20.17, "Time-Stamp Counter," and Table 2-2.		Unique
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.		Shared
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 13.4.4, "Local APIC Status and Location," and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
0	Reserved.	
1	Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
2	Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
3	MCERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
4	Address Parity Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
6: 5	Reserved.	
7	BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
8	Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled.	
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.	
10	MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	
11	Reserved.	
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
13	Reserved	
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes	
15	Reserved.	
17:16	APIC Cluster ID (R/O)	
18	System Bus Frequency (R/O) 0 = 100 MHz. 1 = Reserved.	
19	Reserved.	
21: 20	Symmetric Arbitration ID (R/O)	
26:22	Clock Frequency Ratio (R/O)	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in IA-32 Processor (R/W) See Table 2-2.		Unique
Register Address: 40H, 64	MSR_LASTBRANCH_0	
Last Branch Record 0 (R/W) One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the 'to' address. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."		Unique
Register Address: 41H, 65	MSR_LASTBRANCH_1	
Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2	
Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3	
Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 44H, 68	MSR_LASTBRANCH_4	
Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 45H, 69	MSR_LASTBRANCH_5	
Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 46H, 70	MSR_LASTBRANCH_6	
Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 47H, 71	MSR_LASTBRANCH_7	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Unique
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Unique
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Unique
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Unique
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the scalable bus clock speed.		Shared
2:0	101B: 100 MHz (FSB 400) 001B: 133 MHz (FSB 533) 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 101B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.	
63:3	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Unique
Register Address: 11EH, 281	MSR_BBL_CR_CTL3	
Control Register 3 Used to configure the L2 Cache.		Shared
0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
7:1	Reserved.	
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.	
22:9	Reserved.	
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.	
63:24	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique
0	RIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.	
1	EIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved	
Register Address: 186H, 390	IA32_PERFVTSELO	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2.		Shared
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2 and Section 17.8.2, "Thermal Monitor."		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2 and Section 17.8.2, "Thermal Monitor".		Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Unique
15:0	Reserved.	
16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.	
63:16	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Miscellaneous Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
2:0	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
9:8	Reserved.	
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
12	Reserved.	
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.	Shared
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2. Setting this bit may cause behavior in software that depends on the availability of CPUID leaves greater than 2.	Shared
33:23	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Shared
63:35	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in Table 2-2.		Unique
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 200H, 512	MTRRphysBase0	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Memory Type Range Registers		Unique
Register Address: 201H, 513	MTRRphysMask0	
Memory Type Range Registers		Unique
Register Address: 202H, 514	MTRRphysBase1	
Memory Type Range Registers		Unique
Register Address: 203H, 515	MTRRphysMask1	
Memory Type Range Registers		Unique
Register Address: 204H, 516	MTRRphysBase2	
Memory Type Range Registers		Unique
Register Address: 205H, 517	MTRRphysMask2	
Memory Type Range Registers		Unique
Register Address: 206H, 518	MTRRphysBase3	
Memory Type Range Registers		Unique
Register Address: 207H, 519	MTRRphysMask3	
Memory Type Range Registers		Unique
Register Address: 208H, 520	MTRRphysBase4	
Memory Type Range Registers		Unique
Register Address: 209H, 521	MTRRphysMask4	
Memory Type Range Registers		Unique
Register Address: 20AH, 522	MTRRphysBase5	
Memory Type Range Registers		Unique
Register Address: 20BH, 523	MTRRphysMask5	
Memory Type Range Registers		Unique
Register Address: 20CH, 524	MTRRphysBase6	
Memory Type Range Registers		Unique
Register Address: 20DH, 525	MTRRphysMask6	
Memory Type Range Registers		Unique
Register Address: 20EH, 526	MTRRphysBase7	
Memory Type Range Registers		Unique
Register Address: 20FH, 527	MTRRphysMask7	
Memory Type Range Registers		Unique
Register Address: 250H, 592	MTRRfix64K_00000	
Memory Type Range Registers		Unique
Register Address: 258H, 600	MTRRfix16K_80000	
Memory Type Range Registers		Unique
Register Address: 259H, 601	MTRRfix16K_A0000	
Memory Type Range Registers		Unique

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 268H, 616	MTRRfix4K_C0000	
Memory Type Range Registers		Unique
Register Address: 269H, 617	MTRRfix4K_C8000	
Memory Type Range Registers		Unique
Register Address: 26AH, 618	MTRRfix4K_D0000	
Memory Type Range Registers		Unique
Register Address: 26BH, 619	MTRRfix4K_D8000	
Memory Type Range Registers		Unique
Register Address: 26CH, 620	MTRRfix4K_E0000	
Memory Type Range Registers		Unique
Register Address: 26DH, 621	MTRRfix4K_E8000	
Memory Type Range Registers		Unique
Register Address: 26EH, 622	MTRRfix4K_F0000	
Memory Type Range Registers		Unique
Register Address: 26FH, 623	MTRRfix4K_F8000	
Memory Type Range Registers		Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2 and Section 14.11.2.1, "IA32_MTRR_DEF_TYPE MSR."		Unique
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 408H, 1032	IA32_MC2_CTL	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Unique
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS."		Unique
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 40CH, 1036	MSR_MC4_CTL	
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."		Unique
Register Address: 40DH, 1037	MSR_MC4_STATUS	
See Section 18.3.2.2, "IA32_MCI_STATUS MSRS."		Unique
Register Address: 40EH, 1038	MSR_MC4_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 410H, 1040	IA32_MC3_CTL	
IA32_MC3_CTL	See Section 18.3.2.1, "IA32_MCI_CTL MSRs."	
Register Address: 411H, 1041	IA32_MC3_STATUS	
IA32_MC3_STATUS	See Section 18.3.2.2, "IA32_MCI_STATUS MSRS."	
Register Address: 412H, 1042	MSR_MC3_ADDR	
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 413H, 1043	MSR_MC3_MISC	
Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 414H, 1044	MSR_MC5_CTL	
Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).		Unique
Register Address: 415H, 1045	MSR_MC5_STATUS	
Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.		Unique
Register Address: 416H, 1046	MSR_MC5_ADDR	
Machine Check Error Reporting Register - contains the address of the code or data memory location that produced the machine-check error if the ADDR_V flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 417H, 1047	MSR_MC5_MISC	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISCV flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information." (If CPUID.01H:ECX[5])		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data." (If CPUID.01H:ECX[5])		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0." (If CPUID.01H:ECX[5])		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0." (If CPUID.01H:ECX[5])		Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4." (If CPUID.01H:ECX[5])		Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4." (If CPUID.01H:ECX[5])		Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration." (If CPUID.01H:ECX[5])		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX[5] and IA32_VMX_PROCBASED_CTL2[bit 63])		Unique
Register Address: 600H, 1536	IA32_DS_AREA	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
DS Save Area (R/W) See Table 2-2 and Section 22.6.3.4, "Debug Store (DS) Mechanism."		Unique
31:0	DS Buffer Management Area Linear address of the first byte of the DS buffer management area.	
63:32	Reserved.	
Register Address: C000_0080H	IA32_EFER	
See Table 2-2.		Unique
10:0	Reserved.	
11	Execute Disable Bit Enable	
63:12	Reserved.	

2.21 MSRS IN THE PENTIUM M PROCESSOR

Model-specific registers (MSRs) for the Pentium M processor are similar to those described in Section 2.22 for P6 family processors. The following table describes new MSRs and MSRs whose behavior has changed on the Pentium M processor.

Table 2-67. MSRs in Pentium M Processors

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 0H, 0	P5_MC_ADDR
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER
See Section 20.17, "Time-Stamp Counter," and see Table 2-2.	
Register Address: 17H, 23	IA32_PLATFORM_ID
Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	
Register Address: 2AH, 42	MSR_EBL_CR_POWERON
Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.	
0	Reserved.
1	Data Error Checking Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
2	Response Error Checking Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.
3	MCERR# Drive Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.
4	Address Parity Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.
6:5	Reserved.
7	BINIT# Driver Enable (R) 1 = Enabled; 0 = Disabled. Always 0 on the Pentium M processor.
8	Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled.
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.
10	MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0 on the Pentium M processor.
11	Reserved.
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0 on the Pentium M processor.
13	Reserved.
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes. Always 0 on the Pentium M processor.
15	Reserved.
17:16	APIC Cluster ID (R/O) Always 00B on the Pentium M processor.
18	System Bus Frequency (R/O) 0 = 100 MHz. 1 = Reserved. Always 0 on the Pentium M processor.
19	Reserved.
21:20	Symmetric Arbitration ID (R/O) Always 00B on the Pentium M processor.
26:22	Clock Frequency Ratio (R/O)
Register Address: 40H, 64	MSR_LASTBRANCH_0

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Last Branch Record 0 (R/W) One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the to address. See also: - Last Branch Record Stack TOS at 1C9H. - Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."	
Register Address: 41H, 65	MSR_LASTBRANCH_1
Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 42H, 66	MSR_LASTBRANCH_2
Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 43H, 67	MSR_LASTBRANCH_3
Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 44H, 68	MSR_LASTBRANCH_4
Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 45H, 69	MSR_LASTBRANCH_5
Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 46H, 70	MSR_LASTBRANCH_6
Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 47H, 71	MSR_LASTBRANCH_7
Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 119H, 281	MSR_BBL_CR_CTL
Control Register Used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response.	
63:0	Reserved.
Register Address: 11EH, 281	MSR_BBL_CR_CTL3
Control Register 3 Used to configure the L2 Cache.	
0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.
4:1	Reserved.

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
5	ECC Check Enable (R/O) This bit enables ECC checking on the cache data bus. ECC is always generated on write cycles. 0 = Disabled (default). 1 = Enabled. For the Pentium M processor, ECC checking on the cache data bus is always enabled.
7:6	Reserved.
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
22:9	Reserved.
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.
63:24	Reserved.
Register Address: 179H, 377	IA32_MCG_CAP
Read-only register that provides information about the machine-check architecture of the processor.	
7:0	Count (R/O) Indicates the number of hardware unit error reporting banks available in the processor.
8	IA32_MCG_CTL Present (R/O) 1 = Indicates that the processor implements the MSR_MCG_CTL register found at MSR 17BH. 0 = Not supported.
63:9	Reserved.
Register Address: 17AH, 378	IA32_MCG_STATUS
Global Machine Check Status	
0	RIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
1	EIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
2	MCIP When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
63:3	Reserved.
Register Address: 198H, 408	IA32_PERF_STATUS

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
See Table 2-2.	
Register Address: 199H, 409	IA32_PERF_CTL
See Table 2-2.	
Register Address: 19AH, 410	IA32_CLOCK_MODULATION
Clock Modulation (R/W). See Table 2-2 and Section 17.8.3, "Software Controlled Clock Modulation."	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT
Thermal Interrupt Control (R/W) See Table 2-2 and Section 17.8.2, "Thermal Monitor."	
Register Address: 19CH, 412	IA32_THERM_STATUS
Thermal Monitor Status (R/W) See Table 2-2 and Section 17.8.2, "Thermal Monitor."	
Register Address: 19DH, 413	MSR_THERM2_CTL
Thermal Monitor 2 Control	
15:0	Reserved.
16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
63:16	Reserved.
Register Address: 1A0H, 416	IA32_MISC_ENABLE
Enable Miscellaneous Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.	
2:0	Reserved.
3	Automatic Thermal Control Circuit Enable (R/W) 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows processor clocks to be automatically modulated based on the processor's thermal sensor operation. 0 = Disabled (default). The automatic thermal control circuit enable bit determines if the thermal control circuit (TCC) will be activated when the processor's internal thermal sensor determines the processor is about to exceed its maximum operating temperature. When the TCC is activated and TM1 is enabled, the processors clocks will be forced to a 50% duty cycle. BIOS must enable this feature. The bit should not be confused with the on-demand thermal control circuit enable bit.
6:4	Reserved.
7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
9:8	Reserved.
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.
	Branch Trace Storage Unavailable (R/O) 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported
12	Processor Event Based Sampling Unavailable (R/O) 1 = Processor does not support processor event based sampling (PEBS); 0 = PEBS is supported. The Pentium M processor does not support PEBS.
15:13	Reserved.
16	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled. On the Pentium M processor, this bit may be configured to be read-only.
22:17	Reserved.
23	xTPR Message Disable (R/W) When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority. The default is processor specific.
63:24	Reserved.
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See also: - MSR_LASTBRANCH_0_FROM_IP (at 40H). - Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."	
Register Address: 1D9H, 473	MSR_DEBUGCTLB
Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."	
Register Address: 1DDH, 477	MSR_LER_TO_LIP
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)," and Section 20.16.2, "Last Branch and Last Exception MSRs."	
Register Address: 1DEH, 478	MSR_LER_FROM_LIP

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 20.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)," and Section 20.16.2, "Last Branch and Last Exception MSRs."	
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE
Default Memory Types (R/W) Sets the memory type for the regions of physical memory that are not mapped by the MTRRs. See Section 14.11.2.1, "IA32_MTRR_DEF_TYPE MSR."	
Register Address: 400H, 1024	IA32_MCO_CTL
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 401H, 1025	IA32_MCO_STATUS
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 402H, 1026	IA32_MCO_ADDR
See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 404H, 1028	IA32_MC1_CTL
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 405H, 1029	IA32_MC1_STATUS
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 406H, 1030	IA32_MC1_ADDR
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 408H, 1032	IA32_MC2_CTL
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 409H, 1033	IA32_MC2_STATUS
See Chapter 18.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 40AH, 1034	IA32_MC2_ADDR
See Section 18.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 40CH, 1036	MSR_MC4_CTL
See Section 18.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 40DH, 1037	MSR_MC4_STATUS
See Section 18.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 40EH, 1038	MSR_MC4_ADDR

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 410H, 1040	MSR_MC3_CTL
See Section 18.3.2.1, "IA32_MCI_CTL MSRs."	
Register Address: 411H, 1041	MSR_MC3_STATUS
See Section 18.3.2.2, "IA32_MCI_STATUS MSRs."	
Register Address: 412H, 1042	MSR_MC3_ADDR
See Section 18.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 600H, 1536	IA32_DS_AREA
DS Save Area (R/W) See Table 2-2. Points to the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 22.6.3.4, "Debug Store (DS) Mechanism."	
31:0	DS Buffer Management Area Linear address of the first byte of the DS buffer management area.
63:32	Reserved.

2.22 MSRS IN THE P6 FAMILY PROCESSORS

The following MSRs are defined for the P6 family processors. The MSRs in this table that are shaded are available only in the Pentium II and Pentium III processors. Beginning with the Pentium 4 processor, some of the MSRs in this list have been designated as "architectural" and have had their names changed. See Table 2-2 for a list of the architectural MSRs.

Table 2-68. MSRs in the P6 Family Processors

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 0H, 0	P5_MC_ADDR
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 10H, 16	TSC
See Section 20.17, "Time-Stamp Counter."	
Register Address: 17H, 23	IA32_PLATFORM_ID
Platform ID (R) The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	
49:0	Reserved.

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
52:50	Platform Id (R) Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7
56:53	L2 Cache Latency Read.
59:57	Reserved.
60	Clock Frequency Ratio Read.
63:61	Reserved.
Register Address: 1BH, 27	APIC_BASE
Section 13.4.4, "Local APIC Status and Location."	
7:0	Reserved.
8	Boot Strap Processor Indicator Bit 1 = BSP
10:9	Reserved.
11	APIC Global Enable Bit - Permanent till reset 1 = Enabled. 0 = Disabled.
31:12	APIC Base Address.
63:32	Reserved.
Register Address: 2AH, 42	EBL_CR_POWERON
Processor Hard Power-On Configuration (R/W) Enables and disables processor features, and (R) indicates current processor configuration.	
0	Reserved ¹
1	Data Error Checking Enable (R/W) 1 = Enabled. 0 = Disabled.
2	Response Error Checking Enable FRCERR Observation Enable (R/W) 1 = Enabled. 0 = Disabled.
3	AERR# Drive Enable (R/W) 1 = Enabled. 0 = Disabled.
4	BERR# Enable for Initiator Bus Requests (R/W) 1 = Enabled. 0 = Disabled.

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
5	Reserved.
6	BERR# Driver Enable for Initiator Internal Errors (R/W) 1 = Enabled. 0 = Disabled.
7	BINIT# Driver Enable (R/W) 1 = Enabled. 0 = Disabled.
8	Output Tri-state Enabled (R) 1 = Enabled. 0 = Disabled.
9	Execute BIST (R) 1 = Enabled. 0 = Disabled.
10	AERR# Observation Enabled (R) 1 = Enabled. 0 = Disabled.
11	Reserved.
12	BINIT# Observation Enabled (R) 1 = Enabled. 0 = Disabled.
13	In Order Queue Depth (R) 1 = 1. 0 = 8.
14	1-MByte Power on Reset Vector (R) 1 = 1MByte. 0 = 4GBytes.
15	FRC Mode Enable (R) 1 = Enabled. 0 = Disabled.
17:16	APIC Cluster ID (R)
19:18	System Bus Frequency (R) 00 = 66MHz. 10 = 100Mhz. 01 = 133MHz. 11 = Reserved.
21:20	Symmetric Arbitration ID (R)
25:22	Clock Frequency Ratio (R)
26	Low Power Mode Enable (R/W)
27	Clock Frequency Ratio
63:28	Reserved. ¹
Register Address: 33H, 51	MSR_TEST_CTRL

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Test Control Register	
29:0	Reserved.
30	Streaming Buffer Disable
31	Disable LOCK# Assertion for split locked access.
Register Address: 79H, 121	BIOS_UPDT_TRIG
BIOS Update Trigger Register.	
Register Address: 88H, 136	BBL_CR_D0[63:0]
Chunk 0 data register D[63:0]: used to write to and read from the L2.	
Register Address: 89H, 137	BBL_CR_D1
Chunk 1 data register D[63:0]: used to write to and read from the L2.	
Register Address: 8AH, 138	BBL_CR_D2
Chunk 2 data register D[63:0]: used to write to and read from the L2.	
Register Address: 8BH, 139	BIOS_SIGN/BBL_CR_D3
BIOS Update Signature Register or Chunk 3 data register D[63:0]. Used to write to and read from the L2 depending on the usage model.	
Register Address: C1H, 193	PerfCtr0 (PERFCTR0)
Performance Counter Register See Table 2-2.	
Register Address: C2H, 194	PerfCtr1 (PERFCTR1)
Performance Counter Register See Table 2-2.	
Register Address: FEH, 254	MTRRcap
Memory Type Range Registers	
Register Address: 116H, 278	BBL_CR_ADDR
Address register: used to send specified address (A31-A3) to L2 during cache initialization accesses.	
2:0	Reserved; set to 0.
31:3	Address bits [35:3].
63:32	Reserved.
Register Address: 118H, 280	BBL_CR_DECC
Data ECC register D[7:0]: used to write ECC and read ECC to/from L2.	
Register Address: 119H, 281	BBL_CR_CTL
Control register: used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response.	

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
4:0	L2 Command: 01100 = Data Read w/ LRU update (RLU). 01110 = Tag Read w/ Data Read (TRR). 01111 = Tag Inquire (TI). 00010 = L2 Control Register Read (CR). 00011 = L2 Control Register Write (CW). 010 + MESI encode = Tag Write w/ Data Read (TWR). 111 + MESI encode = Tag Write w/ Data Write (TWW). 100 + MESI encode = Tag Write (TW).
6:5	
7	State to L2
9:8	Reserved.
11:10	Way 0 - 00, Way 1 - 01, Way 2 - 10, Way 3 - 11 Way to L2
13:12	Modified - 11, Exclusive - 10, Shared - 01, Invalid - 00 Way from L2
15:14	State from L2.
16	Reserved.
17	L2 Hit.
18	Reserved.
20:19	User supplied ECC.
21	Processor number: ² Disable = 1. Enable = 0. Reserved.
63:22	Reserved.
Register Address: 11AH, 282	BBL_CR_TRIG
Trigger register: used to initiate a cache configuration accesses access, Write only with Data = 0.	
Register Address: 11BH, 283	BBL_CR_BUSY
Busy register: indicates when a cache configuration accesses L2 command is in progress. D[0] = 1 = BUSY.	
Register Address: 11EH, 286	BBL_CR_CTL3
Control register 3: used to configure the L2 Cache.	
0	L2 Configured (read/write).
4:1	L2 Cache Latency (read/write).
5	ECC Check Enable (read/write).
6	Address Parity Check Enable (read/write).
7	CRTN Parity Check Enable (read/write).
8	L2 Enabled (read/write).

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
10:9	L2 Associativity (read only): 00 = Direct Mapped. 01 = 2 Way. 10 = 4 Way. 11 = Reserved.
12:11	Number of L2 banks (read only).
17:13	Cache size per bank (read/write): 00001 = 256 KBytes. 00010 = 512 KBytes. 00100 = 1 MByte. 01000 = 2 MBytes. 10000 = 4 MBytes.
18	Cache State error checking enable (read/write).
19	Reserved.
22:20	L2 Physical Address Range support: 111 = 64 GBytes. 110 = 32 GBytes. 101 = 16 GBytes. 100 = 8 GBytes. 011 = 4 GBytes. 010 = 2 GBytes. 001 = 1 GByte. 000 = 512 MBytes.
23	L2 Hardware Disable (read only).
24	Reserved.
25	Cache bus fraction (read only).
63:26	Reserved.
Register Address: 174H, 372	SYSENTER_CS_MSR
CS register target for CPL 0 code	
Register Address: 175H, 373	SYSENTER_ESP_MSR
Stack pointer for CPL 0 stack	
Register Address: 176H, 374	SYSENTER_EIP_MSR
CPL 0 code entry point	
Register Address: 179H, 377	MCG_CAP
Machine Check Global Control Register	
Register Address: 17AH, 378	MCG_STATUS
Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.	
Register Address: 17BH, 379	MCG_CTL
Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).	
Register Address: 186H, 390	PerfEvtSel0 (EVNTSEL0)

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Performance Event Select Register 0 (R/W)	
7:0	Event Select Refer to Performance Counter section for a list of event encodings.
15:8	UMASK (Unit Mask) Unit mask register set to 0 to enable all count options.
16	USER Controls the counting of events at Privilege levels of 1, 2, and 3.
17	OS Controls the counting of events at Privilege level of 0.
18	E Occurrence/Duration Mode Select: 1 = Occurrence. 0 = Duration.
19	PC Enabled the signaling of performance counter overflow via BPO pin.
20	INT Enables the signaling of counter overflow via input to APIC: 1 = Enable. 0 = Disable.
22	ENABLE Enables the counting of performance events in both counters: 1 = Enable. 0 = Disable.
23	INV Inverts the result of the CMASK condition: 1 = Inverted. 0 = Non-Inverted.
31:24	CMASK (Counter Mask)
Register Address: 187H, 391	PerfEvtSel1 (EVNTSEL1)
Performance Event Select for Counter 1 (R/W)	
7:0	Event Select Refer to Performance Counter section for a list of event encodings.
15:8	UMASK (Unit Mask) Unit mask register set to 0 to enable all count options.
16	USER Controls the counting of events at Privilege levels of 1, 2, and 3.
17	OS Controls the counting of events at Privilege level of 0.

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
18	E Occurrence/Duration Mode Select: 1 = Occurrence. 0 = Duration.
19	PC Enabled the signaling of performance counter overflow via BP0 pin.
20	INT Enables the signaling of counter overflow via input to APIC. 1 = Enable. 0 = Disable.
23	INV Inverts the result of the CMASK condition. 1 = Inverted. 0 = Non-Inverted.
31:24	CMASK (Counter Mask)
Register Address: 1D9H, 473	DEBUGCTLMR
Enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode.	
0	Enable/Disable Last Branch Records
1	Branch Trap Flag
2	Performance Monitoring/Break Point Pins
3	Performance Monitoring/Break Point Pins
4	Performance Monitoring/Break Point Pins
5	Performance Monitoring/Break Point Pins
6	Enable/Disable Execution Trace Messages
31:7	Reserved.
Register Address: 1DBH, 475	LASTBRANCHFROMIP
32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.	
Register Address: 1DCH, 476	LASTBRANCHTOIP
32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.	
Register Address: 1DDH, 477	LASTINTFROMIP
Last INT from IP	
Register Address: 1DEH, 478	LASTINTTOIP
Last INT to IP	
Register Address: 200H, 512	MTRRphysBase0
Memory Type Range Registers	
Register Address: 201H, 513	MTRRphysMask0
Memory Type Range Registers	

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 202H, 514	MTRRphysBase1
Memory Type Range Registers	
Register Address: 203H, 515	MTRRphysMask1
Memory Type Range Registers	
Register Address: 204H, 516	MTRRphysBase2
Memory Type Range Registers	
Register Address: 205H, 517	MTRRphysMask2
Memory Type Range Registers	
Register Address: 206H, 518	MTRRphysBase3
Memory Type Range Registers	
Register Address: 207H, 519	MTRRphysMask3
Memory Type Range Registers	
Register Address: 208H, 520	MTRRphysBase4
Memory Type Range Registers	
Register Address: 209H, 521	MTRRphysMask4
Memory Type Range Registers	
Register Address: 20AH, 522	MTRRphysBase5
Memory Type Range Registers	
Register Address: 20BH, 523	MTRRphysMask5
Memory Type Range Registers	
Register Address: 20CH, 524	MTRRphysBase6
Memory Type Range Registers	
Register Address: 20DH, 525	MTRRphysMask6
Memory Type Range Registers	
Register Address: 20EH, 526	MTRRphysBase7
Memory Type Range Registers	
Register Address: 20FH, 527	MTRRphysMask7
Memory Type Range Registers	
Register Address: 250H, 592	MTRRfix64K_00000
Memory Type Range Registers	
Register Address: 258H, 600	MTRRfix16K_80000
Memory Type Range Registers	
Register Address: 259H, 601	MTRRfix16K_A0000
Memory Type Range Registers	
Register Address: 268H, 616	MTRRfix4K_C0000
Memory Type Range Registers	
Register Address: 269H, 617	MTRRfix4K_C8000

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Memory Type Range Registers	
Register Address: 26AH, 618	MTRRfix4K_D0000
Memory Type Range Registers	
Register Address: 26BH, 619	MTRRfix4K_D8000
Memory Type Range Registers	
Register Address: 26CH, 620	MTRRfix4K_E0000
Memory Type Range Registers	
Register Address: 26DH, 621	MTRRfix4K_E8000
Memory Type Range Registers	
Register Address: 26EH, 622	MTRRfix4K_F0000
Memory Type Range Registers	
Register Address: 26FH, 623	MTRRfix4K_F8000
Memory Type Range Registers	
Register Address: 2FFH, 767	MTRRdefType
Memory Type Range Registers	
2:0	Default memory type
10	Fixed MTRR enable
11	MTRR Enable
Register Address: 400H, 1024	MCO_CTL
Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).	
Register Address: 401H, 1025	MCO_STATUS
Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.	
15:0	MC_STATUS_MCACOD
31:16	MC_STATUS_MSCOD
57	MC_STATUS_DAM
58	MC_STATUS_ADDRV
59	MC_STATUS_MISCV
60	MC_STATUS_EN. (Note: For MCO_STATUS only, this bit is hardcoded to 1.)
61	MC_STATUS_UC
62	MC_STATUS_O
63	MC_STATUS_V
Register Address: 402H, 1026	MCO_ADDR
Register Address: 403H, 1027	MCO_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 404H, 1028	MC1_CTL

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 405H, 1029	MC1_STATUS
Bit definitions same as MC0_STATUS.	
Register Address: 406H, 1030	MC1_ADDR
Register Address: 407H, 1031	MC1_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 408H, 1032	MC2_CTL
Register Address: 409H, 1033	MC2_STATUS
Bit definitions same as MC0_STATUS.	
Register Address: 40AH, 1034	MC2_ADDR
Register Address: 40BH, 1035	MC2_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 40CH, 1036	MC4_CTL
Register Address: 40DH, 1037	MC4_STATUS
Bit definitions same as MC0_STATUS, except bits 0, 4, 57, and 61 are hardcoded to 1.	
Register Address: 40EH, 1038	MC4_ADDR
Defined in MCA architecture but not implemented in P6 Family processors.	
Register Address: 40FH, 1039	MC4_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 410H, 1040	MC3_CTL
Register Address: 411H, 1041	MC3_STATUS
Bit definitions same as MC0_STATUS.	
Register Address: 412H, 1042	MC3_ADDR
Register Address: 413H, 1043	MC3_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
NOTES 1. Bit 0 of this register has been redefined several times, and is no longer used in P6 family processors. 2. The processor number feature may be disabled by setting bit 21 of the BBL_CR_CTL MSR (model-specific register address 119h) to "1". Once set, bit 21 of the BBL_CR_CTL may not be cleared. This bit is write-once. The processor number feature will be disabled until the processor is reset. 3. The Pentium III processor will prevent FSB frequency overclocking with a new shutdown mechanism. If the FSB frequency selected is greater than the internal FSB frequency the processor will shutdown. If the FSB selected is less than the internal FSB frequency the BIOS may choose to use bit 11 to implement its own shutdown policy.	

2.23 MSRS IN PENTIUM PROCESSORS

The following MSRs are defined for the Pentium processors. The P5_MC_ADDR, P5_MC_TYPE, and TSC MSRs (named IA32_P5_MC_ADDR, IA32_P5_MC_TYPE, and IA32_TIME_STAMP_COUNTER in the Pentium 4 processor) are architectural; that is, code that accesses these registers will run on Pentium 4 and P6 family processors without generating exceptions (see Section 2.1, "Architectural MSRs"). The CESR, CTR0, and CTR1 MSRs are unique to Pentium processors; code that accesses these registers will generate exceptions on Pentium 4 and P6 family processors.

Table 2-69. MSRs in the Pentium Processor

Register Address: Hex, Decimal	Register Name
Register Information	
Register Address: 0H, 0	P5_MC_ADDR
See Section 18.11.2, "Pentium Processor Machine-Check Exception Handling."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 18.11.2, "Pentium Processor Machine-Check Exception Handling."	
Register Address: 10H, 16	TSC
See Section 20.17, "Time-Stamp Counter."	
Register Address: 11H, 17	CESR
See Section 22.6.9.1, "Control and Event Select Register (CESR)."	
Register Address: 12H, 18	CTR0
Section 22.6.9.3, "Events Counted."	
Register Address: 13H, 19	CTR1
Section 22.6.9.3, "Events Counted."	

