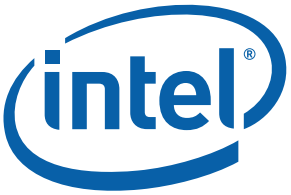


Intel® 64 and IA-32 Architectures Software Developer's Manual

Documentation Changes

December 2021

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: 252046-068

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.

Contents

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

§

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

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 1, Volume 1
2	Updates to Chapter 4, Volume 1
3	Updates to Chapter 5, Volume 1
4	Updates to Chapter 8, Volume 1
5	Updates to Chapter 13, Volume 1
6	Updates to Chapter 1, Volume 2A
7	Updates to Chapter 3, Volume 2A
8	Updates to Chapter 4, Volume 2B
9	Updates to Chapter 5, Volume 2C
10	Updates to Chapter 1, Volume 3A
11	Updates to Chapter 4, Volume 3A
12	Updates to Chapter 6, Volume 3A
13	Updates to Chapter 8, Volume 3A
14	Updates to Chapter 10, Volume 3A
15	Updates to Chapter 14, Volume 3B
16	Updates to Chapter 16, Volume 3B
17	Updates to Chapter 17, Volume 3B
18	New Chapter 18, Volume 3B
19	Updates to Chapter 19, Volume 3B
20	Updates to Chapter 24, Volume 3C
21	Updates to Chapter 26, Volume 3C
22	Updates to Chapter 27, Volume 3C
23	Updates to Chapter 28, Volume 3C
24	Updates to Chapter 32, Volume 3C
25	Updates to Chapter 37, Volume 3D
26	Updates to Appendix A, Volume 3D

Documentation Changes(Sheet 2 of 2)

No.	DOCUMENTATION CHANGES
27	Updates to Appendix B, Volume 3D
28	Updates to Chapter 1, Volume 4
29	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 1, Volume 1

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

Changes to this chapter: Updated section 1.1 "Intel® 64 and IA-32 Processors Covered in this Manual" to add the 12th generation Intel® Core™ processor. Additional minor corrections in chapter.

CHAPTER 1
ABOUT THIS MANUAL

The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture* (order number 253665) is part of a set that describes the architecture and programming environment of Intel® 64 and IA-32 architecture processors. Other volumes in this set are:

- The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*: *Instruction Set Reference* (order numbers 253666, 253667, 326018 and 334569).
- The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A, 3B, 3C & 3D: System Programming Guide* (order numbers 253668, 253669, 326019 and 332831).
- The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers* (order number 335592).

The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1,* describes the basic architecture and programming environment of Intel 64 and IA-32 processors. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*, describe the instruction set of the processor and the opcode structure. These volumes apply to application programmers and to programmers who write operating systems or executives. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A, 3B, 3C & 3D,* describe the operating-system support environment of Intel 64 and IA-32 processors. These volumes target operating-system and BIOS designers. In addition, the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*, addresses the programming environment for classes of software that host operating systems. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4,* describes the model-specific registers of Intel 64 and IA-32 processors.

1.1 INTEL® 64 AND IA-32 PROCESSORS COVERED IN THIS MANUAL

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™2 Duo processor
- Intel® Core™2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series
- Intel® Core™2 Extreme processor X7000 and X6800 series
- Intel® Core™2 Extreme processor QX6000 series
- Intel® Xeon® processor 7100 series

- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel Atom® processor family
- Intel Atom® processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are built from 45 nm and 32 nm processes
- Intel® Core™ i7 processor
- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Core™ i7-3930K processor
- 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series
- Intel® Xeon® processor E3-1200 product family
- Intel® Xeon® processor E5-2400/1400 product family
- Intel® Xeon® processor E5-4600/2600/1600 product family
- 3rd generation Intel® Core™ processors
- Intel® Xeon® processor E3-1200 v2 product family
- Intel® Xeon® processor E5-2400/1400 v2 product families
- Intel® Xeon® processor E5-4600/2600/1600 v2 product families
- Intel® Xeon® processor E7-8800/4800/2800 v2 product families
- 4th generation Intel® Core™ processors
- The Intel® Core™ M processor family
- Intel® Core™ i7-59xx Processor Extreme Edition
- Intel® Core™ i7-49xx Processor Extreme Edition
- Intel® Xeon® processor E3-1200 v3 product family
- Intel® Xeon® processor E5-2600/1600 v3 product families
- 5th generation Intel® Core™ processors
- Intel® Xeon® processor D-1500 product family
- Intel® Xeon® processor E5 v4 family
- Intel Atom® processor X7-Z8000 and X5-Z8000 series
- Intel Atom® processor Z3400 series
- Intel Atom® processor Z3500 series
- 6th generation Intel® Core™ processors
- Intel® Xeon® processor E3-1500m v5 product family
- 7th generation Intel® Core™ processors
- Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series
- Intel® Xeon® Processor Scalable Family
- 8th generation Intel® Core™ processors
- Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series
- Intel® Xeon® E processors
- 9th generation Intel® Core™ processors
- 2nd generation Intel® Xeon® Processor Scalable Family

- 10th generation Intel® Core™ processors
- 11th generation Intel® Core™ processors
- 3rd generation Intel® Xeon® Processor Scalable Family
- 12th generation Intel® Core™ processors

P6 family processors are IA-32 processors based on the P6 family microarchitecture. This includes the Pentium® Pro, Pentium® II, Pentium® III, and Pentium® III Xeon® processors.

The Pentium® 4, Pentium® D, and Pentium® processor Extreme Editions are based on the Intel NetBurst® microarchitecture. Most early Intel® Xeon® processors are based on the Intel NetBurst® microarchitecture. Intel Xeon processor 5000, 7100 series are based on the Intel NetBurst® microarchitecture.

The Intel® Core™ Duo, Intel® Core™ Solo and dual-core Intel® Xeon® processor LV are based on an improved Pentium® M processor microarchitecture.

The Intel® Xeon® processor 3000, 3200, 5100, 5300, 7200, and 7300 series, Intel® Pentium® dual-core, Intel® Core™2 Duo, Intel® Core™2 Quad, and Intel® Core™2 Extreme processors are based on Intel® Core™ microarchitecture.

The Intel® Xeon® processor 5200, 5400, 7400 series, Intel® Core™2 Quad processor Q9000 series, and Intel® Core™2 Extreme processors QX9000, X9000 series, Intel® Core™2 processor E8000 series are based on Enhanced Intel® Core™ microarchitecture.

The Intel Atom® processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are based on the Intel Atom® microarchitecture and supports Intel 64 architecture.

P6 family, Pentium® M, Intel® Core™ Solo, Intel® Core™ Duo processors, dual-core Intel® Xeon® processor LV, and early generations of Pentium 4 and Intel Xeon processors support IA-32 architecture. The Intel® Atom™ processor Z5xx series support IA-32 architecture.

The Intel® Xeon® processor 3000, 3200, 5000, 5100, 5200, 5300, 5400, 7100, 7200, 7300, 7400 series, Intel® Core™2 Duo, Intel® Core™2 Extreme, Intel® Core™2 Quad processors, Pentium® D processors, Pentium® Dual-Core processor, newer generations of Pentium 4 and Intel Xeon processor family support Intel® 64 architecture.

The Intel® Core™ i7 processor and Intel® Xeon® processor 3400, 5500, 7500 series are based on 45 nm Nehalem microarchitecture. Westmere microarchitecture is a 32 nm version of the Nehalem microarchitecture. Intel® Xeon® processor 5600 series, Intel Xeon processor E7 and various Intel Core i7, i5, i3 processors are based on the Westmere microarchitecture. These processors support Intel 64 architecture.

The Intel® Xeon® processor E5 family, Intel® Xeon® processor E3-1200 family, Intel® Xeon® processor E7-8800/4800/2800 product families, Intel® Core™ i7-3930K processor, and 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Sandy Bridge microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E7-8800/4800/2800 v2 product families, Intel® Xeon® processor E3-1200 v2 product family and 3rd generation Intel® Core™ processors are based on the Ivy Bridge microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E5-4600/2600/1600 v2 product families, Intel® Xeon® processor E5-2400/1400 v2 product families and Intel® Core™ i7-49xx Processor Extreme Edition are based on the Ivy Bridge-E microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E3-1200 v3 product family and 4th Generation Intel® Core™ processors are based on the Haswell microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E5-2600/1600 v3 product families and the Intel® Core™ i7-59xx Processor Extreme Edition are based on the Haswell-E microarchitecture and support Intel 64 architecture.

The Intel Atom® processor Z8000 series is based on the Airmont microarchitecture.

The Intel Atom® processor Z3400 series and the Intel Atom® processor Z3500 series are based on the Silvermont microarchitecture.

The Intel® Core™ M processor family, 5th generation Intel® Core™ processors, Intel® Xeon® processor D-1500 product family and the Intel® Xeon® processor E5 v4 family are based on the Broadwell microarchitecture and support Intel 64 architecture.

The Intel® Xeon® Processor Scalable Family, Intel® Xeon® processor E3-1500m v5 product family and 6th generation Intel® Core™ processors are based on the Skylake microarchitecture and support Intel 64 architecture.

The 7th generation Intel® Core™ processors are based on the Kaby Lake microarchitecture and support Intel 64 architecture.

The Intel Atom® processor C series, the Intel Atom® processor X series, the Intel® Pentium® processor J series, the Intel® Celeron® processor J series, and the Intel® Celeron® processor N series are based on the Goldmont microarchitecture.

The Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series is based on the Knights Landing microarchitecture and supports Intel 64 architecture.

The Intel® Pentium® Silver processor series, the Intel® Celeron® processor J series, and the Intel® Celeron® processor N series are based on the Goldmont Plus microarchitecture.

The 8th generation Intel® Core™ processors, 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on the Coffee Lake microarchitecture and support Intel 64 architecture.

The Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series is based on the Knights Mill microarchitecture and supports Intel 64 architecture.

The 2nd generation Intel® Xeon® Processor Scalable Family is based on the Cascade Lake product and supports Intel 64 architecture.

Some 10th generation Intel® Core™ processors are based on the Ice Lake microarchitecture, and some are based on the Comet Lake microarchitecture; both support Intel 64 architecture.

Some 11th generation Intel® Core™ processors are based on the Tiger Lake microarchitecture, and some are based on the Rocket Lake microarchitecture; both support Intel 64 architecture.

Some 3rd generation Intel® Xeon® Processor Scalable Family processors are based on the Cooper Lake product, and some are based on the Ice Lake microarchitecture; both support Intel 64 architecture.

The 12th generation Intel® Core™ processors are based on the Alder Lake performance hybrid architecture and support Intel 64 architecture.

IA-32 architecture is the instruction set architecture and programming environment for Intel's 32-bit microprocessors. Intel® 64 architecture is the instruction set architecture and programming environment which is the superset of Intel’s 32-bit and 64-bit architectures. It is compatible with the IA-32 architecture.

1.2 OVERVIEW OF VOLUME 1: BASIC ARCHITECTURE

A description of this manual’s content follows:

Chapter 1 — About This Manual. Gives an overview of all five volumes of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual.* It also describes the notational conventions in these manuals and lists related Intel manuals and documentation of interest to programmers and hardware designers.

Chapter 2 — Intel® 64 and IA-32 Architectures. Introduces the Intel 64 and IA-32 architectures along with the families of Intel processors that are based on these architectures. It also gives an overview of the common features found in these processors and brief history of the Intel 64 and IA-32 architectures.

Chapter 3 — Basic Execution Environment. Introduces the models of memory organization and describes the register set used by applications.

Chapter 4 — Data Types. Describes the data types and addressing modes recognized by the processor; provides an overview of real numbers and floating-point formats and of floating-point exceptions.

Chapter 5 — Instruction Set Summary. Lists all Intel 64 and IA-32 instructions, divided into technology groups.

Chapter 6 — Procedure Calls, Interrupts, and Exceptions. Describes the procedure stack and mechanisms provided for making procedure calls and for servicing interrupts and exceptions.

Chapter 7 — Programming with General-Purpose Instructions. Describes basic load and store, program control, arithmetic, and string instructions that operate on basic data types, general-purpose and segment registers; also describes system instructions that are executed in protected mode.

Chapter 8 — Programming with the x87 FPU. Describes the x87 floating-point unit (FPU), including floating-point registers and data types; gives an overview of the floating-point instruction set and describes the processor's floating-point exception conditions.

Chapter 9 — Programming with Intel® MMX™ Technology. Describes Intel MMX technology, including MMX registers and data types; also provides an overview of the MMX instruction set.

Chapter 10 — Programming with Intel® Streaming SIMD Extensions (Intel® SSE). Describes SSE extensions, including XMM registers, the MXCSR register, and packed single-precision floating-point data types; provides an overview of the SSE instruction set and gives guidelines for writing code that accesses the SSE extensions.

Chapter 11 — Programming with Intel® Streaming SIMD Extensions 2 (Intel® SSE2). Describes SSE2 extensions, including XMM registers and packed double-precision floating-point data types; provides an overview of the SSE2 instruction set and gives guidelines for writing code that accesses SSE2 extensions. This chapter also describes SIMD floating-point exceptions that can be generated with SSE and SSE2 instructions. It also provides general guidelines for incorporating support for SSE and SSE2 extensions into operating system and applications code.

Chapter 12 — Programming with Intel® Streaming SIMD Extensions 3 (Intel® SSE3), Supplemental Streaming SIMD Extensions 3 (SSSE3), Intel® Streaming SIMD Extensions 4 (Intel® SSE4) and Intel® AES New Instructions (Intel® AES-NI). Provides an overview of the SSE3 instruction set, Supplemental SSE3, SSE4, AESNI instructions, and guidelines for writing code that access these extensions.

Chapter 13 — Managing State Using the XSAVE Feature Set. Describes the XSAVE feature set instructions and explains how software can enable the XSAVE feature set and XSAVE-enabled features.

Chapter 14 — Programming with AVX, FMA and AVX2. Provides an overview of the Intel® AVX instruction set, FMA and Intel AVX2 extensions and gives guidelines for writing code that access these extensions.

Chapter 15 — Programming with Intel® AVX-512. Provides an overview of the Intel® AVX-512 instruction set extensions and gives guidelines for writing code that access these extensions.

Chapter 16 — Programming with Intel Transactional Synchronization Extensions. Describes the instruction extensions that support lock elision techniques to improve the performance of multi-threaded software with contended locks.

Chapter 17 — Intel® Memory Protection Extensions. Provides an overview of the Intel® Memory Protection Extensions and gives guidelines for writing code that access these extensions.

Chapter 18 — Control-flow Enforcement Technology. Provides an overview of the Control-flow Enforcement Technology (CET) and gives guidelines for writing code that access these extensions.

Chapter 19 — Input/Output. Describes the processor’s I/O mechanism, including I/O port addressing, I/O instructions, and I/O protection mechanisms.

Chapter 20 — Processor Identification and Feature Determination. Describes how to determine the CPU type and features available in the processor.

Appendix A — EFLAGS Cross-Reference. Summarizes how the IA-32 instructions affect the flags in the EFLAGS register.

Appendix B — EFLAGS Condition Codes. Summarizes how conditional jump, move, and ‘byte set on condition code’ instructions use condition code flags (OF, CF, ZF, SF, and PF) in the EFLAGS register.

Appendix C — Floating-Point Exceptions Summary. Summarizes exceptions raised by the x87 FPU floating-point and SSE/SSE2/SSE3 floating-point instructions.

Appendix D — Guidelines for Writing SIMD Floating-Point Exception Handlers. Gives guidelines for writing exception handlers for exceptions generated by SSE/SSE2/SSE3 floating-point instructions.

1.3 NOTATIONAL CONVENTIONS

This manual uses specific notation for data-structure formats, for symbolic representation of instructions, and for hexadecimal and binary numbers. This notation is described below.

1.3.1 Bit and Byte Order

In illustrations of data structures in memory, smaller addresses appear toward the bottom of the figure; addresses increase toward the top. Bit positions are numbered from right to left. The numerical value of a set bit is equal to two raised to the power of the bit position. Intel 64 and IA-32 processors are "little endian" machines; this means the bytes of a word are numbered starting from the least significant byte. See Figure 1-1.

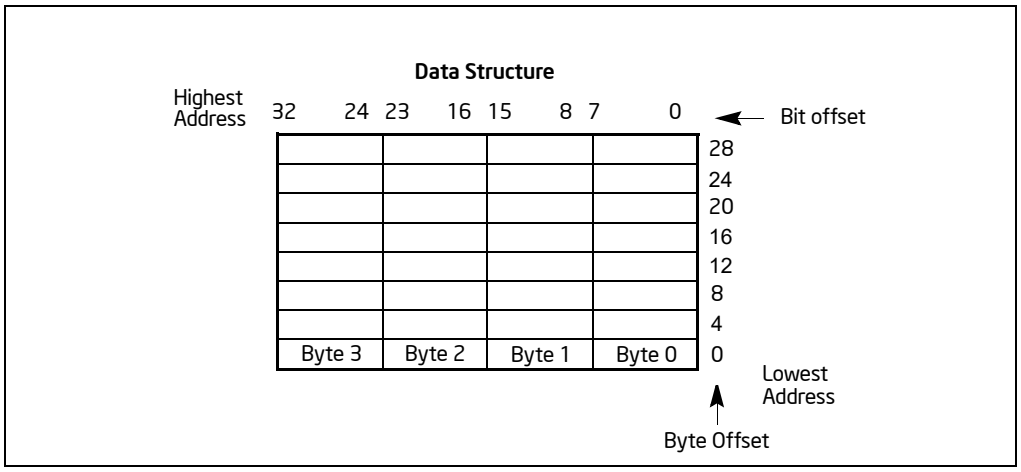


Figure 1-1. Bit and Byte Order

1.3.2 Reserved Bits and Software Compatibility

In many register and memory layout descriptions, certain bits are marked as **reserved**. When bits are marked as reserved, it is essential for compatibility with future processors that software treat these bits as having a future, though unknown, effect. The behavior of reserved bits should be regarded as not only undefined, but unpredictable.

Software should follow these guidelines in dealing with reserved bits:

- Do not depend on the states of any reserved bits when testing the values of registers that contain such bits. Mask out the reserved bits before testing.
- Do not depend on the states of any reserved bits when storing to memory or to a register.
- Do not depend on the ability to retain information written into any reserved bits.
- When loading a register, always load the reserved bits with the values indicated in the documentation, if any, or reload them with values previously read from the same register.

NOTE

Avoid any software dependence upon the state of reserved bits in Intel 64 and IA-32 registers. Depending upon the values of reserved register bits will make software dependent upon the unspecified manner in which the processor handles these bits. Programs that depend upon reserved values risk incompatibility with future processors.

1.3.2.1 Instruction Operands

When instructions are represented symbolically, a subset of the IA-32 assembly language is used. In this subset, an instruction has the following format:

```
label: mnemonic argument1, argument2, argument3
```

where:

- A **label** is an identifier which is followed by a colon.
- A **mnemonic** is a reserved name for a class of instruction opcodes which have the same function.

- The operands **argument1**, **argument2**, and **argument3** are optional. There may be from zero to three operands, depending on the opcode. When present, they take the form of either literals or identifiers for data items. Operand identifiers are either reserved names of registers or are assumed to be assigned to data items declared in another part of the program (which may not be shown in the example).

When two operands are present in an arithmetic or logical instruction, the right operand is the source and the left operand is the destination.

For example:

```
LOADREG: MOV EAX, SUBTOTAL
```

In this example, LOADREG is a label, MOV is the mnemonic identifier of an opcode, EAX is the destination operand, and SUBTOTAL is the source operand. Some assembly languages put the source and destination in reverse order.

1.3.3 Hexadecimal and Binary Numbers

Base 16 (hexadecimal) numbers are represented by a string of hexadecimal digits followed by the character H (for example, 0F82EH). A hexadecimal digit is a character from the following set: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Base 2 (binary) numbers are represented by a string of 1s and 0s, sometimes followed by the character B (for example, 1010B). The "B" designation is only used in situations where confusion as to the type of number might arise.

1.3.4 Segmented Addressing

The processor uses byte addressing. This means memory is organized and accessed as a sequence of bytes. Whether one or more bytes are being accessed, a byte address is used to locate the byte or bytes memory. The range of memory that can be addressed is called an **address space**.

The processor also supports segmented addressing. This is a form of addressing where a program may have many independent address spaces, called **segments**. For example, a program can keep its code (instructions) and stack in separate segments. Code addresses would always refer to the code space, and stack addresses would always refer to the stack space. The following notation is used to specify a byte address within a segment:

```
Segment-register:Byte-address
```

For example, the following segment address identifies the byte at address FF79H in the segment pointed by the DS register:

```
DS:FF79H
```

The following segment address identifies an instruction address in the code segment. The CS register points to the code segment and the EIP register contains the address of the instruction.

```
CS:EIP
```

1.3.5 A New Syntax for CPUID, CR, and MSR Values

Obtain feature flags, status, and system information by using the CPUID instruction, by checking control register bits, and by reading model-specific registers. We are moving toward a new syntax to represent this information. See Figure 1-2.

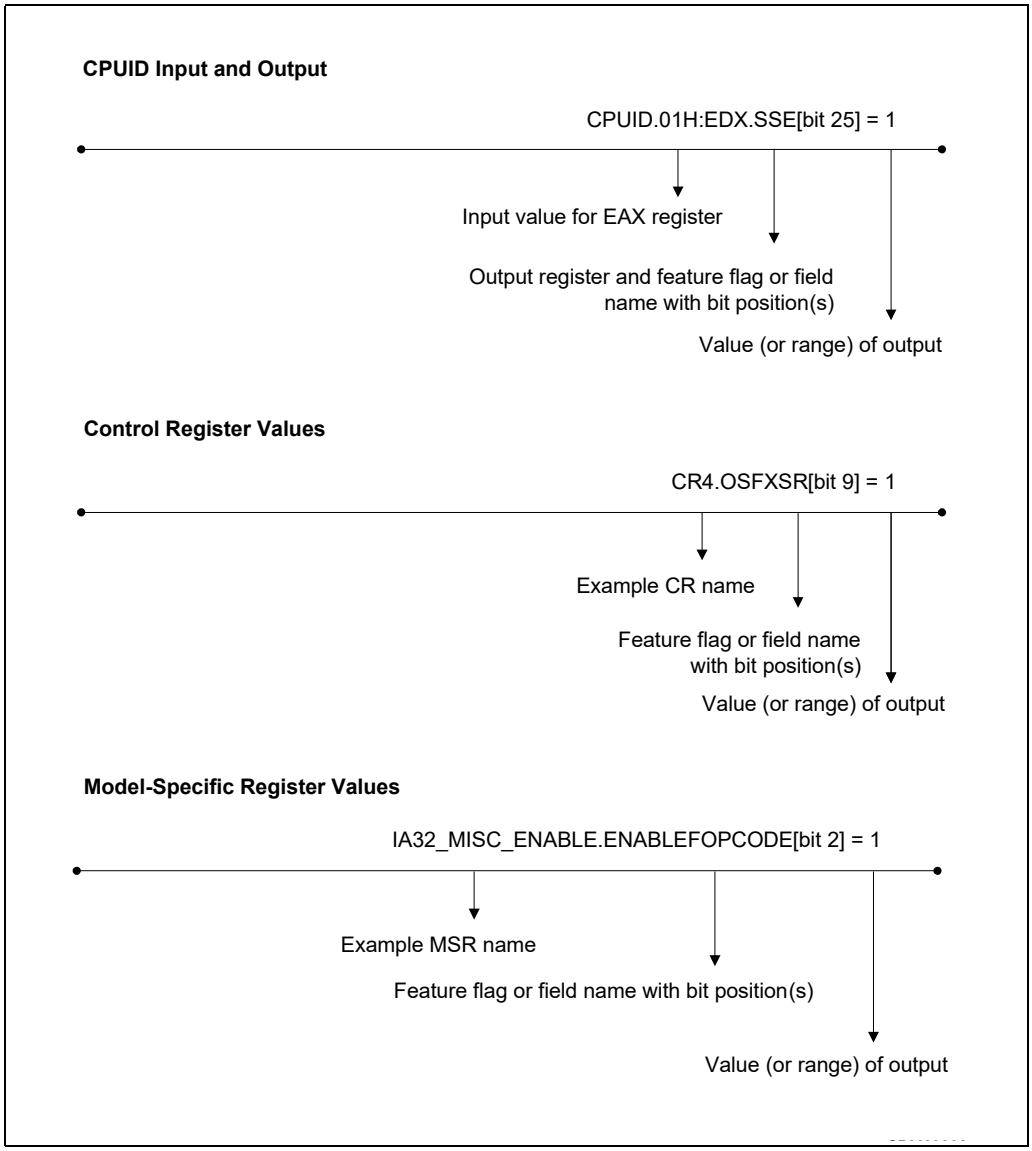


Figure 1-2. Syntax for CPUID, CR, and MSR Data Presentation

1.3.6 Exceptions

An exception is an event that typically occurs when an instruction causes an error. For example, an attempt to divide by zero generates an exception. However, some exceptions, such as breakpoints, occur under other conditions. Some types of exceptions may provide error codes. An error code reports additional information about the error. An example of the notation used to show an exception and error code is shown below:

#PF(fault code)

This example refers to a page-fault exception under conditions where an error code naming a type of fault is reported. Under some conditions, exceptions that produce error codes may not be able to report an accurate code. In this case, the error code is zero, as shown below for a general-protection exception:

#GP(0)

1.4 RELATED LITERATURE

Literature related to Intel 64 and IA-32 processors is listed and viewable on-line at:

https://software.intel.com/en-us/articles/intel-sdm

See also:

- The latest security information on Intel® products:
  https://www.intel.com/content/www/us/en/security-center/default.html
- Software developer resources, guidance and insights for security advisories:
  https://software.intel.com/security-software-guidance/
- The data sheet for a particular Intel 64 or IA-32 processor
- The specification update for a particular Intel 64 or IA-32 processor
- Intel® C++ Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Fortran Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Software Development Tools:
  https://software.intel.com/en-us/intel-sdp-home
- Intel® 64 and IA-32 Architectures Software Developer's Manual (in one, four or ten volumes):
  https://software.intel.com/en-us/articles/intel-sdm
- Intel® 64 and IA-32 Architectures Optimization Reference Manual:
  https://software.intel.com/en-us/articles/intel-sdm#optimization
- Intel® Trusted Execution Technology Measured Launched Environment Programming Guide:
  http://www.intel.com/content/www/us/en/software-developers/intel-txt-software-development-guide.html
- Intel® Software Guard Extensions (Intel® SGX) Information
  https://software.intel.com/en-us/isa-extensions/intel-sgx
- Developing Multi-threaded Applications: A Platform Consistent Approach:
  https://software.intel.com/sites/default/files/article/147714/51534-developing-multithreaded-applications.pdf
- Using Spin-Loops on Intel® Pentium® 4 Processor and Intel® Xeon® Processor:
  https://software.intel.com/sites/default/files/22/30/25602
- Performance Monitoring Unit Sharing Guide
  http://software.intel.com/file/30388

Literature related to select features in future Intel processors are available at:

- Intel® Architecture Instruction Set Extensions Programming Reference
  https://software.intel.com/en-us/isa-extensions

More relevant links are:

- Intel® Developer Zone:
  https://software.intel.com/en-us
- Developer centers:
  http://www.intel.com/content/www/us/en/hardware-developers/developer-centers.html
- Processor support general link:
  http://www.intel.com/support/processors/
- Intel® Hyper-Threading Technology (Intel® HT Technology):
  http://www.intel.com/technology/platform-technology/hyper-threading/index.htm

2. Updates to Chapter 4, Volume 1

Change bars and green text show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

Changes to this chapter: Updates to Table 4-2, "Length, Precision, and Range of Floating-Point Data Types" in Section 4.2.2, "Floating-Point Data Types".

CHAPTER 4
DATA TYPES

This chapter introduces data types defined for the Intel 64 and IA-32 architectures. A section at the end of this chapter describes the real-number and floating-point concepts used in x87 FPU, SSE, SSE2, SSE3, SSSE3, SSE4 and Intel AVX extensions.

4.1 FUNDAMENTAL DATA TYPES

The fundamental data types are bytes, words, doublewords, quadwords, and double quadwords (see Figure 4-1). A byte is eight bits, a word is 2 bytes (16 bits), a doubleword is 4 bytes (32 bits), a quadword is 8 bytes (64 bits), and a double quadword is 16 bytes (128 bits). A subset of the IA-32 architecture instructions operates on these fundamental data types without any additional operand typing.

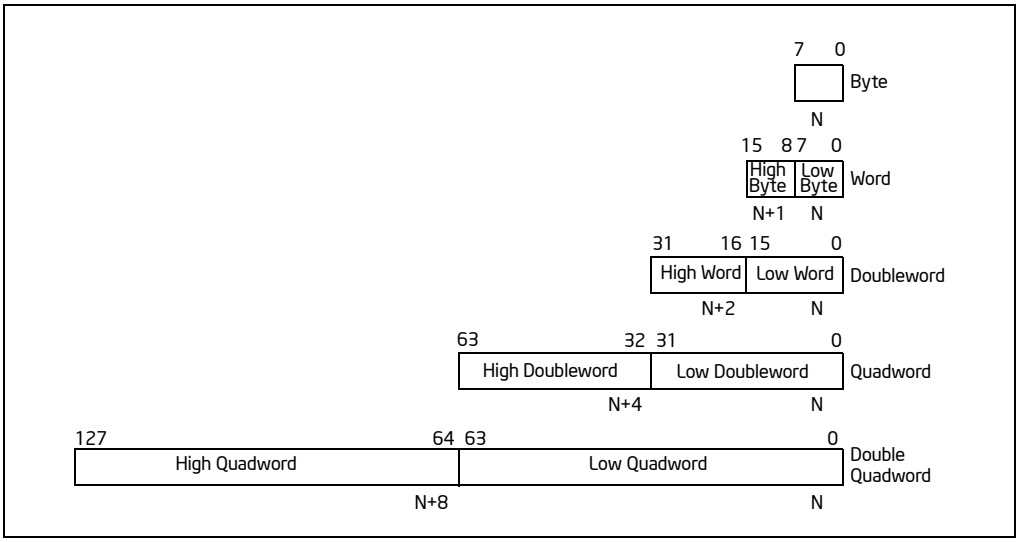


Figure 4-1. Fundamental Data Types

The quadword data type was introduced into the IA-32 architecture in the Intel486 processor; the double quadword data type was introduced in the Pentium III processor with the SSE extensions.

Figure 4-2 shows the byte order of each of the fundamental data types when referenced as operands in memory. The low byte (bits 0 through 7) of each data type occupies the lowest address in memory and that address is also the address of the operand.

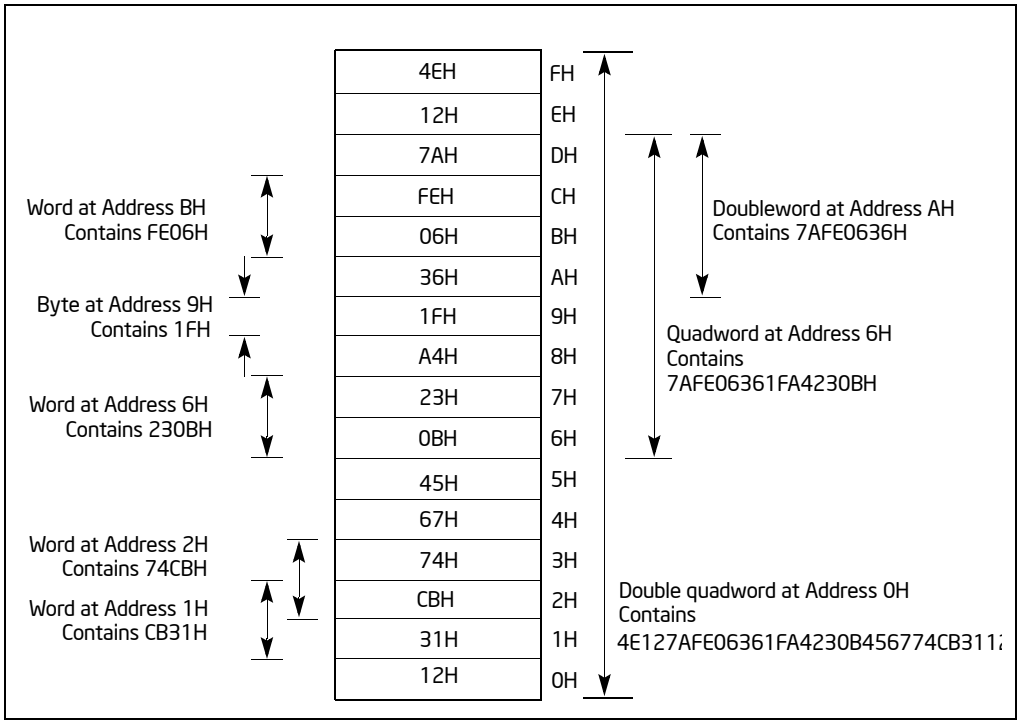


Figure 4-2. Bytes, Words, Doublewords, Quadwords, and Double Quadwords in Memory

4.1.1 Alignment of Words, Doublewords, Quadwords, and Double Quadwords

Words, doublewords, and quadwords do not need to be aligned in memory on natural boundaries. The natural boundaries for words, double words, and quadwords are even-numbered addresses, addresses evenly divisible by four, and addresses evenly divisible by eight, respectively. However, to improve the performance of programs, data structures (especially stacks) should be aligned on natural boundaries whenever possible. The reason for this is that the processor requires two memory accesses to make an unaligned memory access; aligned accesses require only one memory access. A word or doubleword operand that crosses a 4-byte boundary or a quadword operand that crosses an 8-byte boundary is considered unaligned and requires two separate memory bus cycles for access.

Some instructions that operate on double quadwords require memory operands to be aligned on a natural boundary. These instructions generate a general-protection exception (#GP) if an unaligned operand is specified. A natural boundary for a double quadword is any address evenly divisible by 16. Other instructions that operate on double quadwords permit unaligned access (without generating a general-protection exception). However, additional memory bus cycles are required to access unaligned data from memory.

4.2 NUMERIC DATA TYPES

Although bytes, words, and doublewords are fundamental data types, some instructions support additional interpretations of these data types to allow operations to be performed on numeric data types (signed and unsigned integers, and floating-point numbers). Single-precision (32-bit) floating-point and double-precision (64-bit) floating-point data types are supported across all generations of SSE extensions and Intel AVX extensions. Half-precision (16-bit) floating-point data type is supported only with F16C extensions (VCVTPH2PS, VCVTPS2PH). See Figure 4-3.

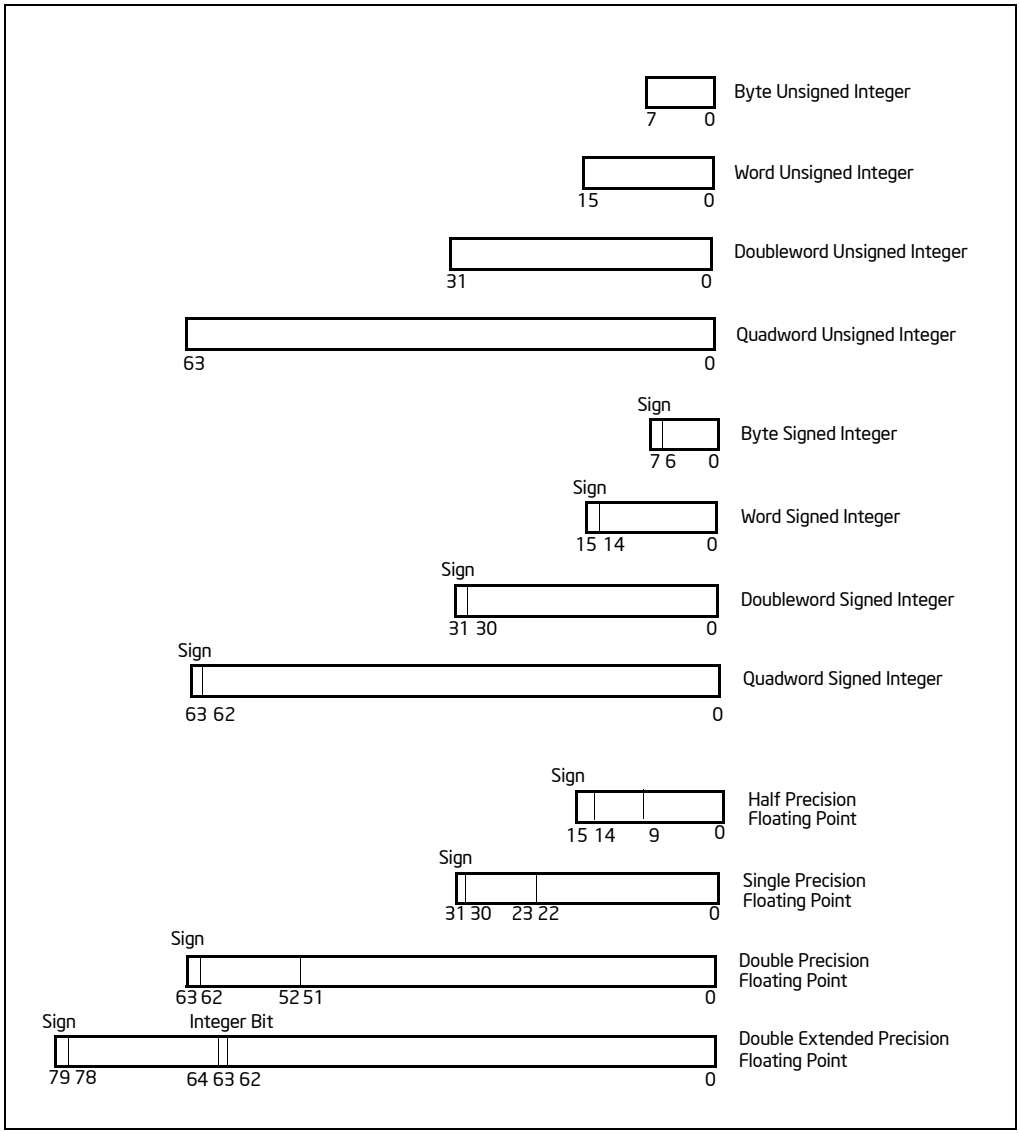


Figure 4-3. Numeric Data Types

4.2.1 Integers

The Intel 64 and IA-32 architectures define two types of integers: unsigned and signed. Unsigned integers are ordinary binary values ranging from 0 to the maximum positive number that can be encoded in the selected operand size. Signed integers are two's complement binary values that can be used to represent both positive and negative integer values.

Some integer instructions (such as the ADD, SUB, PADDB, and PSUBB instructions) operate on either unsigned or signed integer operands. Other integer instructions (such as IMUL, MUL, IDIV, DIV, FIADD, and FISUB) operate on only one integer type.

The following sections describe the encodings and ranges of the two types of integers.

4.2.1.1 Unsigned Integers

Unsigned integers are unsigned binary numbers contained in a byte, word, doubleword, and quadword. Their values range from 0 to 255 for an unsigned byte integer, from 0 to 65,535 for an unsigned word integer, from 0

to $2^{32} - 1$ for an unsigned doubleword integer, and from 0 to $2^{64} - 1$ for an unsigned quadword integer. Unsigned integers are sometimes referred to as **ordinals**.

4.2.1.2 Signed Integers

Signed integers are signed binary numbers held in a byte, word, doubleword, or quadword. All operations on signed integers assume a two's complement representation. The sign bit is located in bit 7 in a byte integer, bit 15 in a word integer, bit 31 in a doubleword integer, and bit 63 in a quadword integer (see the signed integer encodings in Table 4-1).

Table 4-1. Signed Integer Encodings

Class		Two's Complement Encoding	
		Sign	
Positive	Largest	0	11..11
	.	.	.
	.	.	.
	Smallest	0	00..01
Zero		0	00..00
Negative	Smallest	1	11..11
	.	.	.
	.	.	.
	Largest	1	00..00
Integer indefinite		1	00..00
		Signed Byte Integer:	← 7 bits →
		Signed Word Integer:	← 15 bits →
		Signed Doubleword Integer:	← 31 bits →
		Signed Quadword Integer:	← 63 bits →

The sign bit is set for negative integers and cleared for positive integers and zero. Integer values range from −128 to +127 for a byte integer, from −32,768 to +32,767 for a word integer, from -2^{31} to $+2^{31} - 1$ for a doubleword integer, and from -2^{63} to $+2^{63} - 1$ for a quadword integer.

When storing integer values in memory, word integers are stored in 2 consecutive bytes; doubleword integers are stored in 4 consecutive bytes; and quadword integers are stored in 8 consecutive bytes.

The integer indefinite is a special value that is sometimes returned by the x87 FPU when operating on integer values. For more information, see Section 8.2.1, "Indefinites."

4.2.2 Floating-Point Data Types

The IA-32 architecture defines and operates on three floating-point data types: single-precision floating-point, double-precision floating-point, and double-extended precision floating-point (see Figure 4-3). The data formats for these data types correspond directly to formats specified in the IEEE Standard 754 for Binary Floating-Point Arithmetic.

Half-precision (16-bit) floating-point data type is supported only for conversion operation with single-precision floating data using F16C extensions (VCVTPH2PS, VCVTPS2PH).

Table 4-2 gives the length, precision, and approximate normalized range that can be represented by each of these data types. Denormal values are also supported in each of these types.

Table 4-2. Length, Precision, and Range of Floating-Point Data Types

Data Type	Length (Bits)	Precision (Bits)	Approximate Normalized Range	
			Binary	Decimal
Half Precision	16	11	2^{-14} to 2^{16}	6.10×10^{-5} to 6.55×10^{4}
Single Precision	32	24	2^{-126} to 2^{128}	1.18×10^{-38} to 3.40×10^{38}
Double Precision	64	53	2^{-1022} to 2^{1024}	2.23×10^{-308} to 1.80×10^{308}
Double-Extended Precision	80	64	2^{-16382} to 2^{16384}	3.36×10^{-4932} to 1.19×10^{4932}

NOTE

Section 4.8, "Real Numbers and Floating-Point Formats," gives an overview of the IEEE Standard 754 floating-point formats and defines the terms integer bit, QNaN, SNaN, and denormal value.

Table 4-3 shows the floating-point encodings for zeros, denormalized finite numbers, normalized finite numbers, infinites, and NaNs for each of the three floating-point data types. It also gives the format for the QNaN floating-point indefinite value. (See Section 4.8.3.7, "QNaN Floating-Point Indefinite," for a discussion of the use of the QNaN floating-point indefinite value.)

For the single-precision and double-precision formats, only the fraction part of the significand is encoded. The integer is assumed to be 1 for all numbers except 0 and denormalized finite numbers. For the double extended-precision format, the integer is contained in bit 63, and the most-significant fraction bit is bit 62. Here, the integer is explicitly set to 1 for normalized numbers, infinities, and NaNs, and to 0 for zero and denormalized numbers.

Table 4-3. Floating-Point Number and NaN Encodings

Class		Sign	Biased Exponent	Significand	
				Integer[1]	Fraction
Positive	+∞	0	11..11	1	00..00
	+Normals	0 . . 0	11..10 . . 00..01	1 . . 1	11..11 . . 00..00
	+Denormals	0 . . 0	00..00 . . 00..00	0 . . 0	11.11 . . 00..01
	+Zero	0	00..00	0	00..00
Negative	–Zero	1	00..00	0	00..00
	–Denormals	1 . . 1	00..00 . . 00..00	0 . . 0	00..01 . . 11..11
	–Normals	1 . . 1	00..01 . . 11..10	1 . . 1	00..00 . . 11..11
	-∞	1	11..11	1	00..00

Table 4-3. Floating-Point Number and NaN Encodings (Contd.)

Class		Sign	Biased Exponent	Significand	
				Integer[1]	Fraction
NaNs	SNaN	X	11..11	1	0X..XX[2]
	QNaN	X	11..11	1	1X..XX
	QNaN Floating-Point Indefinite	1	11..11	1	10..00
	Half-Precision Single-Precision: Double-Precision: Double Extended-Precision:		← 5Bits → ← 8 Bits → ← 11 Bits → ← 15 Bits →		← 10 Bits → ← 23 Bits → ← 52 Bits → ← 63 Bits →

NOTES:

1. Integer bit is implied and not stored for single-precision and double-precision formats.
2. The fraction for SNaN encodings must be non-zero with the most-significant bit 0.

The exponent of each floating-point data type is encoded in biased format; see Section 4.8.2.2, "Biased Exponent." The biasing constant is 15 for the half-precision format, 127 for the single-precision format, 1023 for the double-precision format, and 16,383 for the double extended-precision format.

When storing floating-point values in memory, half-precision values are stored in 2 consecutive bytes in memory; single-precision values are stored in 4 consecutive bytes in memory; double-precision values are stored in 8 consecutive bytes; and double extended-precision values are stored in 10 consecutive bytes.

The single-precision and double-precision floating-point data types are operated on by x87 FPU, and SSE/SSE2/SSE3/SSE4.1 and Intel AVX instructions. The double-extended-precision floating-point format is only operated on by the x87 FPU. See Section 11.6.8, "Compatibility of SIMD and x87 FPU Floating-Point Data Types," for a discussion of the compatibility of single-precision and double-precision floating-point data types between the x87 FPU and SSE/SSE2/SSE3 extensions.

4.3 POINTER DATA TYPES

Pointers are addresses of locations in memory.

In non-64-bit modes, the architecture defines two types of pointers: a **near pointer** and a **far pointer**. A near pointer is a 32-bit (or 16-bit) offset (also called an **effective address**) within a segment. Near pointers are used for all memory references in a flat memory model or for references in a segmented model where the identity of the segment being accessed is implied.

A far pointer is a logical address, consisting of a 16-bit segment selector and a 32-bit (or 16-bit) offset. Far pointers are used for memory references in a segmented memory model where the identity of a segment being accessed must be specified explicitly. Near and far pointers with 32-bit offsets are shown in Figure 4-4.

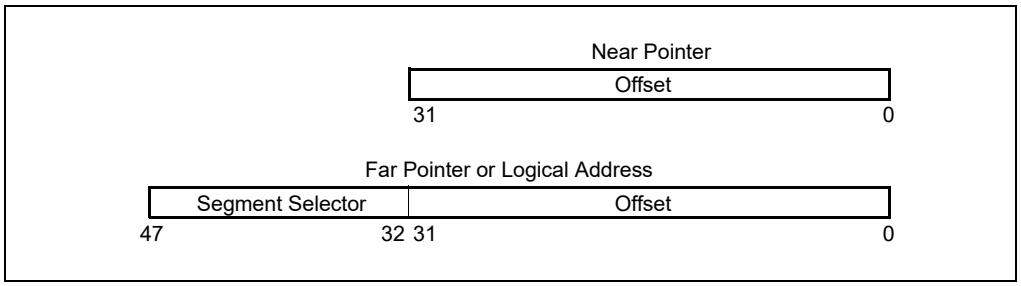


Figure 4-4. Pointer Data Types

4.3.1 Pointer Data Types in 64-Bit Mode

In 64-bit mode (a sub-mode of IA-32e mode), a **near pointer** is 64 bits. This equates to an effective address. **Far pointers** in 64-bit mode can be one of three forms:

- 16-bit segment selector, 16-bit offset if the operand size is 32 bits
- 16-bit segment selector, 32-bit offset if the operand size is 32 bits
- 16-bit segment selector, 64-bit offset if the operand size is 64 bits

See Figure 4-5.

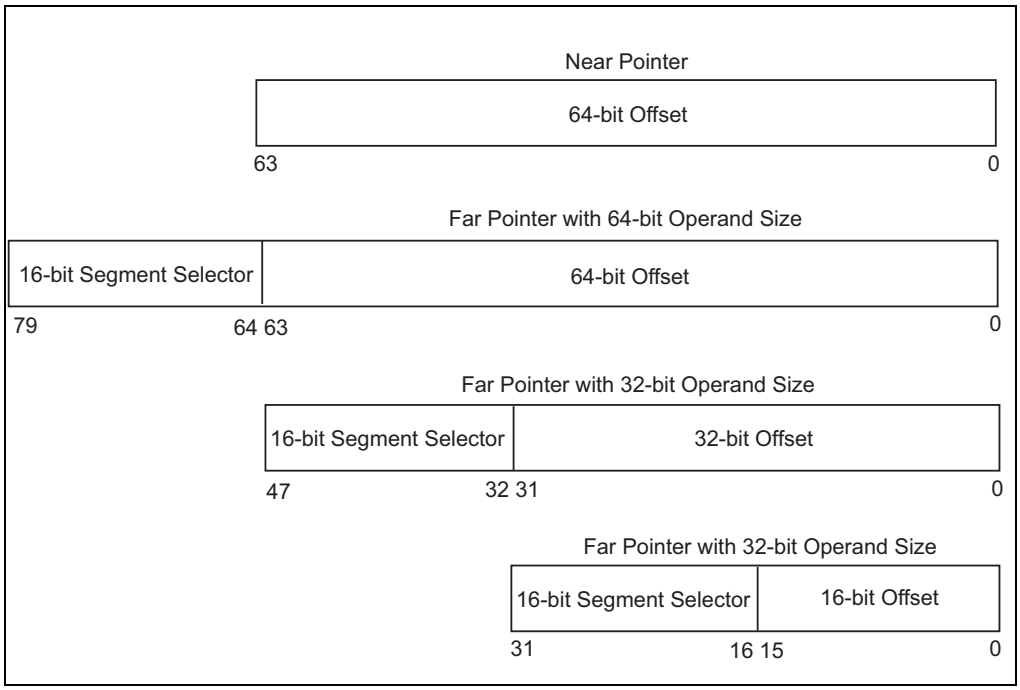


Figure 4-5. Pointers in 64-Bit Mode

4.4 BIT FIELD DATA TYPE

A **bit field** (see Figure 4-6) is a contiguous sequence of bits. It can begin at any bit position of any byte in memory and can contain up to 32 bits.

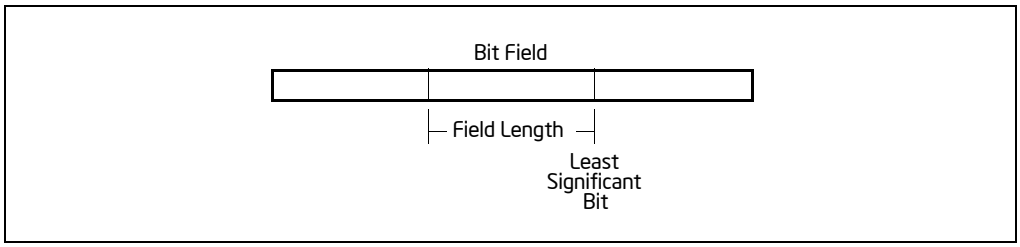


Figure 4-6. Bit Field Data Type

4.5 STRING DATA TYPES

Strings are continuous sequences of bits, bytes, words, or doublewords. A **bit string** can begin at any bit position of any byte and can contain up to $2^{32} - 1$ bits. A **byte string** can contain bytes, words, or doublewords and can range from zero to $2^{32} - 1$ bytes (4 GBytes).

4.6 PACKED SIMD DATA TYPES

Intel 64 and IA-32 architectures define and operate on a set of 64-bit and 128-bit packed data type for use in SIMD operations. These data types consist of fundamental data types (packed bytes, words, doublewords, and quadwords) and numeric interpretations of fundamental types for use in packed integer and packed floating-point operations.

4.6.1 64-Bit SIMD Packed Data Types

The 64-bit packed SIMD data types were introduced into the IA-32 architecture in the Intel MMX technology. They are operated on in MMX registers. The fundamental 64-bit packed data types are packed bytes, packed words, and packed doublewords (see Figure 4-7). When performing numeric SIMD operations on these data types, these data types are interpreted as containing byte, word, or doubleword integer values.

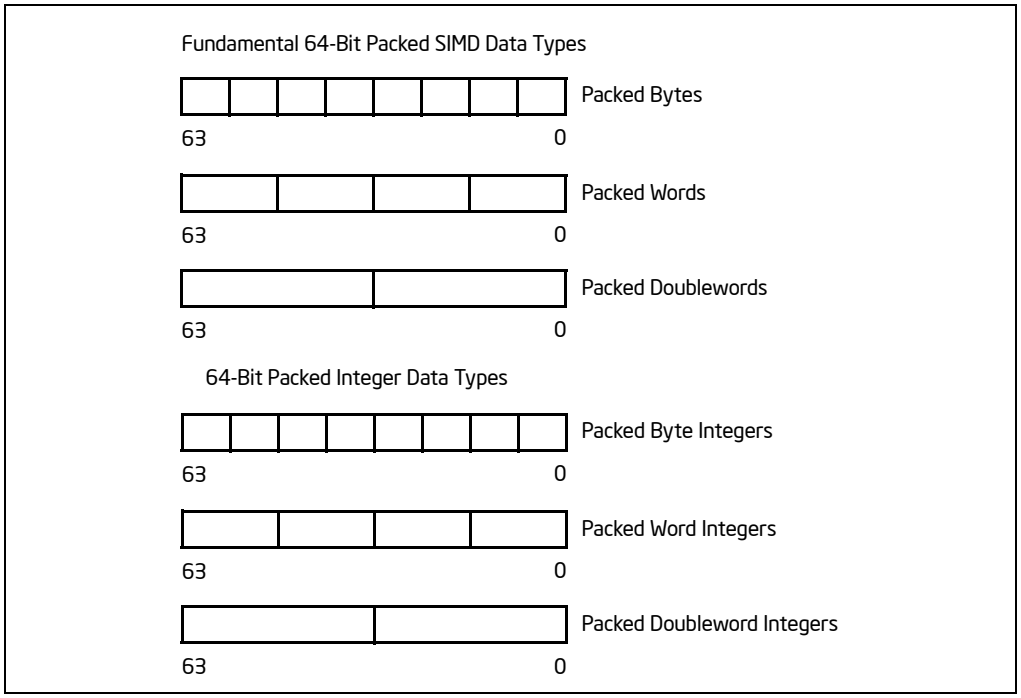


Figure 4-7. 64-Bit Packed SIMD Data Types

4.6.2 128-Bit Packed SIMD Data Types

The 128-bit packed SIMD data types were introduced into the IA-32 architecture in the SSE extensions and used with SSE2, SSE3 and SSSE3 extensions. They are operated on primarily in the 128-bit XMM registers and memory. The fundamental 128-bit packed data types are packed bytes, packed words, packed doublewords, and packed quadwords (see Figure 4-8). When performing SIMD operations on these fundamental data types in XMM registers, these data types are interpreted as containing packed or scalar single-precision floating-point or double-precision floating-point values, or as containing packed byte, word, doubleword, or quadword integer values.

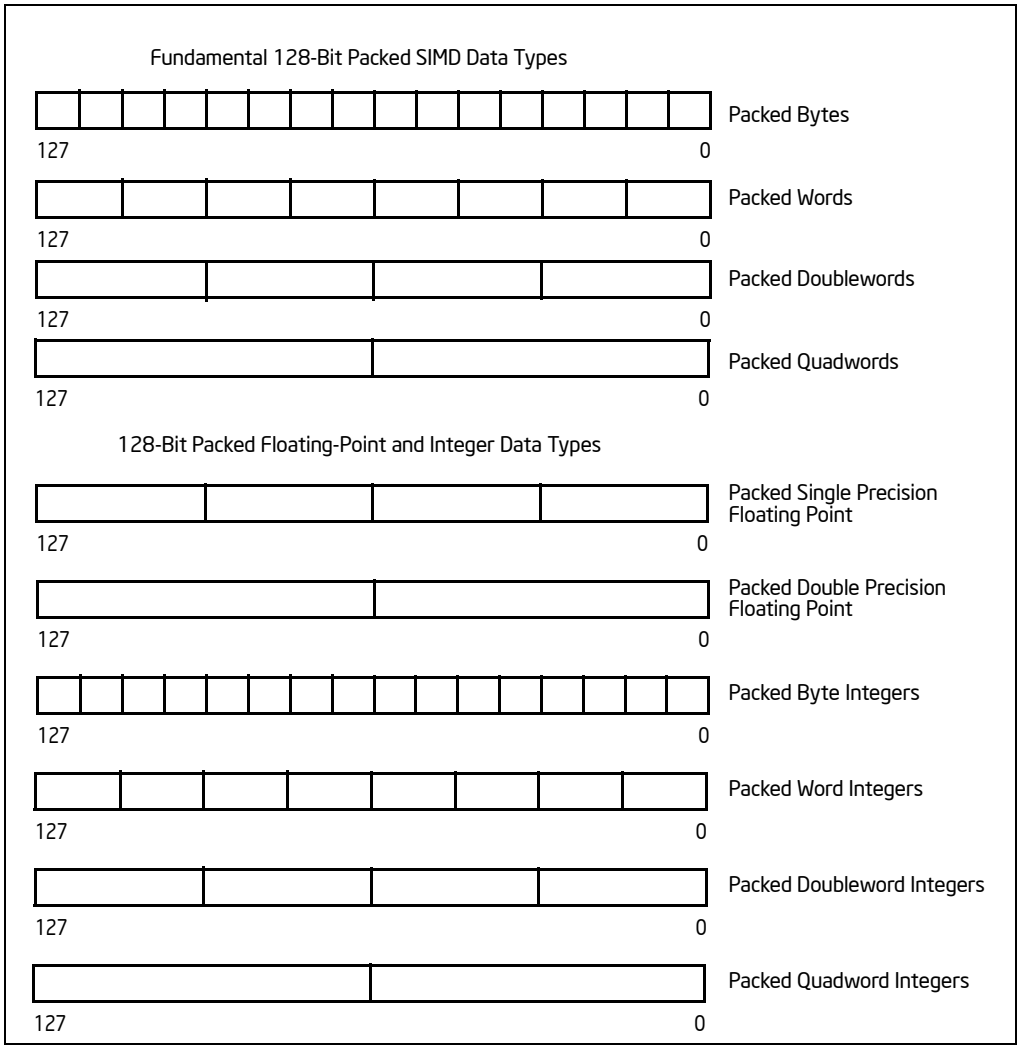


Figure 4-8. 128-Bit Packed SIMD Data Types

4.7 BCD AND PACKED BCD INTEGERS

Binary-coded decimal integers (BCD integers) are unsigned 4-bit integers with valid values ranging from 0 to 9. IA-32 architecture defines operations on BCD integers located in one or more general-purpose registers or in one or more x87 FPU registers (see Figure 4-9).

Figure 4-9. BCD Data Types

When operating on BCD integers in general-purpose registers, the BCD values can be unpacked (one BCD digit per byte) or packed (two BCD digits per byte). The value of an unpacked BCD integer is the binary value of the low half-byte (bits 0 through 3). The high half-byte (bits 4 through 7) can be any value during addition and subtraction, but must be zero during multiplication and division. Packed BCD integers allow two BCD digits to be contained in one byte. Here, the digit in the high half-byte is more significant than the digit in the low half-byte.

When operating on BCD integers in x87 FPU data registers, BCD values are packed in an 80-bit format and referred to as decimal integers. In this format, the first 9 bytes hold 18 BCD digits, 2 digits per byte. The least-significant digit is contained in the lower half-byte of byte 0 and the most-significant digit is contained in the upper half-byte of byte 9. The most significant bit of byte 10 contains the sign bit (0 = positive and 1 = negative; bits 0 through 6 of byte 10 are don't care bits). Negative decimal integers are not stored in two's complement form; they are distinguished from positive decimal integers only by the sign bit. The range of decimal integers that can be encoded in this format is $-10^{18}+1$ to $10^{18}-1$.

The decimal integer format exists in memory only. When a decimal integer is loaded in an x87 FPU data register, it is automatically converted to the double-extended-precision floating-point format. All decimal integers are exactly representable in double extended-precision format.

Table 4-4 gives the possible encodings of value in the decimal integer data type.

Table 4-4. Packed Decimal Integer Encodings

Class	Sign		Magnitude					
			digit	digit	digit	digit	...	digit
Positive								
Largest	0	0000000	1001	1001	1001	1001	...	1001
	.	.			.			
	.	.			.			
Smallest	0	0000000	0000	0000	0000	0000	...	0001
Zero	0	0000000	0000	0000	0000	0000	...	0000
Negative								
Zero	1	0000000	0000	0000	0000	0000	...	0000
Smallest	1	0000000	0000	0000	0000	0000	...	0001
	.	.			.			
	.	.			.			
Largest	1	0000000	1001	1001	1001	1001	...	1001

Table 4-4. Packed Decimal Integer Encodings (Contd.)

Class	Sign		Magnitude					
			digit	digit	digit	digit	...	digit
Packed BCD Integer Indefinite	1	1111111	1111	1111	1100	0000	...	0000
	← 1 byte →		← 9 bytes →					

The packed BCD integer indefinite encoding (FFFFC000000000000000H) is stored by the FBSTP instruction in response to a masked floating-point invalid-operation exception. Attempting to load this value with the FBLD instruction produces an undefined result.

4.8 REAL NUMBERS AND FLOATING-POINT FORMATS

This section describes how real numbers are represented in floating-point format in x87 FPU and SSE/SSE2/SSE3/SSE4.1 and Intel AVX floating-point instructions. It also introduces terms such as normalized numbers, denormalized numbers, biased exponents, signed zeros, and NaNs. Readers who are already familiar with floating-point processing techniques and the IEEE Standard 754 for Binary Floating-Point Arithmetic may wish to skip this section.

4.8.1 Real Number System

As shown in Figure 4-10, the real-number system comprises the continuum of real numbers from minus infinity (– ∞) to plus infinity (+ ∞).

Because the size and number of registers that any computer can have is limited, only a subset of the real-number continuum can be used in real-number (floating-point) calculations. As shown at the bottom of Figure 4-10, the subset of real numbers that the IA-32 architecture supports represents an approximation of the real number system. The range and precision of this real-number subset is determined by the IEEE Standard 754 floating-point formats.

4.8.2 Floating-Point Format

To increase the speed and efficiency of real-number computations, computers and microprocessors typically represent real numbers in a binary floating-point format. In this format, a real number has three parts: a sign, a significand, and an exponent (see Figure 4-11).

The sign is a binary value that indicates whether the number is positive (0) or negative (1). The significand has two parts: a 1-bit binary integer (also referred to as the J-bit) and a binary fraction. The integer-bit is often not represented, but instead is an implied value. The exponent is a binary integer that represents the base-2 power by which the significand is multiplied.

Table 4-5 shows how the real number 178.125 (in ordinary decimal format) is stored in IEEE Standard 754 floating-point format. The table lists a progression of real number notations that leads to the single-precision, 32-bit floating-point format. In this format, the significand is normalized (see Section 4.8.2.1, "Normalized Numbers") and the exponent is biased (see Section 4.8.2.2, "Biased Exponent"). For the single-precision floating-point format, the biasing constant is +127.

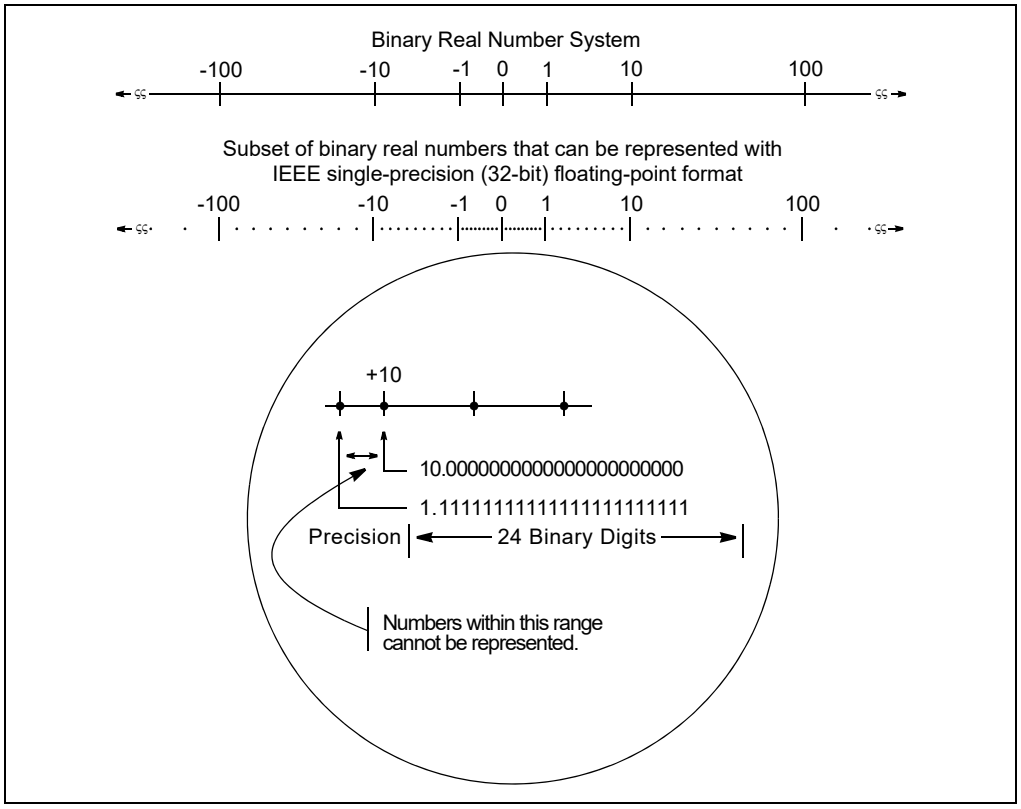


Figure 4-10. Binary Real Number System

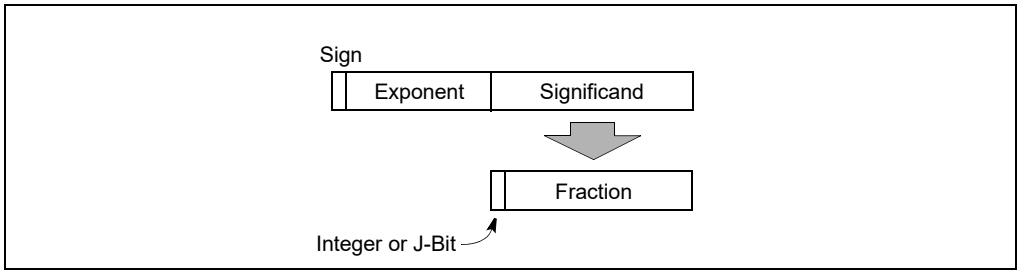


Figure 4-11. Binary Floating-Point Format

Table 4-5. Real and Floating-Point Number Notation

Notation	Value		
Ordinary Decimal	178.125		
Scientific Decimal	$1.78125E_{10}2$		
Scientific Binary	$1.0110010001E_{2}111$		
Scientific Binary (Biased Exponent)	$1.0110010001E_{2}10000110$		
IEEE Single-Precision Format	Sign	Biased Exponent	Normalized Significand
	0	10000110	01100100010000000000000 1. (Implied)

4.8.2.1 Normalized Numbers

In most cases, floating-point numbers are encoded in normalized form. This means that except for zero, the significand is always made up of an integer of 1 and the following fraction:

1.fff...ff

For values less than 1, leading zeros are eliminated. (For each leading zero eliminated, the exponent is decremented by one.)

Representing numbers in normalized form maximizes the number of significant digits that can be accommodated in a significand of a given width. To summarize, a normalized real number consists of a normalized significand that represents a real number between 1 and 2 and an exponent that specifies the number's binary point.

4.8.2.2 Biased Exponent

In the IA-32 architecture, the exponents of floating-point numbers are encoded in a biased form. This means that a constant is added to the actual exponent so that the biased exponent is always a positive number. The value of the biasing constant depends on the number of bits available for representing exponents in the floating-point format being used. The biasing constant is chosen so that the smallest normalized number can be reciprocated without overflow.

See Section 4.2.2, "Floating-Point Data Types," for a list of the biasing constants that the IA-32 architecture uses for the various sizes of floating-point data-types.

4.8.3 Real Number and Non-number Encodings

A variety of real numbers and special values can be encoded in the IEEE Standard 754 floating-point format. These numbers and values are generally divided into the following classes:

- Signed zeros
- Denormalized finite numbers
- Normalized finite numbers
- Signed infinities
- NaNs
- Indefinite numbers

(The term NaN stands for "Not a Number.")

Figure 4-12 shows how the encodings for these numbers and non-numbers fit into the real number continuum. The encodings shown here are for the IEEE single-precision floating-point format. The term "S" indicates the sign bit, "E" the biased exponent, and "Sig" the significand. The exponent values are given in decimal. The integer bit is shown for the significands, even though the integer bit is implied in single-precision floating-point format.

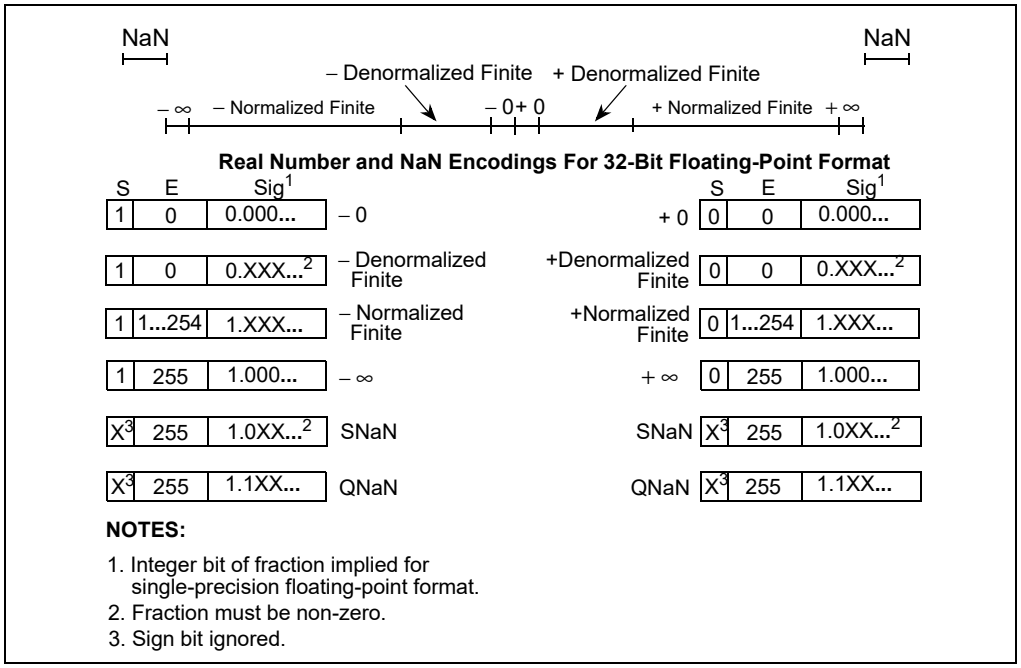


Figure 4-12. Real Numbers and NaNs

An IA-32 processor can operate on and/or return any of these values, depending on the type of computation being performed. The following sections describe these number and non-number classes.

4.8.3.1 Signed Zeros

Zero can be represented as a +0 or a –0 depending on the sign bit. Both encodings are equal in value. The sign of a zero result depends on the operation being performed and the rounding mode being used. Signed zeros have been provided to aid in implementing interval arithmetic. The sign of a zero may indicate the direction from which underflow occurred, or it may indicate the sign of an ∞ that has been reciprocated.

4.8.3.2 Normalized and Denormalized Finite Numbers

Non-zero, finite numbers are divided into two classes: normalized and denormalized. The normalized finite numbers comprise all the non-zero finite values that can be encoded in a normalized real number format between zero and ∞. In the single-precision floating-point format shown in Figure 4-12, this group of numbers includes all the numbers with biased exponents ranging from 1 to 254_{10} (unbiased, the exponent range is from -126_{10} to $+127_{10}$).

When floating-point numbers become very close to zero, the normalized-number format can no longer be used to represent the numbers. This is because the range of the exponent is not large enough to compensate for shifting the binary point to the right to eliminate leading zeros.

When the biased exponent is zero, smaller numbers can only be represented by making the integer bit (and perhaps other leading bits) of the significand zero. The numbers in this range are called **denormalized** numbers. The use of leading zeros with denormalized numbers allows smaller numbers to be represented. However, this denormalization may cause a loss of precision (the number of significant bits is reduced by the leading zeros).

When performing normalized floating-point computations, an IA-32 processor normally operates on normalized numbers and produces normalized numbers as results. Denormalized numbers represent an **underflow** condition. The exact conditions are specified in Section 4.9.1.5, "Numeric Underflow Exception (#U)."

A denormalized number is computed through a technique called gradual underflow. Table 4-6 gives an example of gradual underflow in the denormalization process. Here the single-precision format is being used, so the minimum exponent (unbiased) is -126_{10}. The true result in this example requires an exponent of -129_{10} in order to have a

normalized number. Since -129_{10} is beyond the allowable exponent range, the result is denormalized by inserting leading zeros until the minimum exponent of -126_{10} is reached.

Table 4-6. Denormalization Process

Operation	Sign	Exponent*	Significand
True Result	0	–129	1.01011100000...00
Denormalize	0	–128	0.10101110000...00
Denormalize	0	–127	0.01010111000...00
Denormalize	0	–126	0.00101011100...00
Denormal Result	0	–126	0.00101011100...00

* Expressed as an unbiased, decimal number.

In the extreme case, all the significant bits are shifted out to the right by leading zeros, creating a zero result.

The Intel 64 and IA-32 architectures deal with denormal values in the following ways:

- It avoids creating denormals by normalizing numbers whenever possible.
- It provides the floating-point underflow exception to permit programmers to detect cases when denormals are created.
- It provides the floating-point denormal-operand exception to permit procedures or programs to detect when denormals are being used as source operands for computations.

4.8.3.3 Signed Infinities

The two infinities, + ∞ and – ∞, represent the maximum positive and negative real numbers, respectively, that can be represented in the floating-point format. Infinity is always represented by a significand of 1.00...00 (the integer bit may be implied) and the maximum biased exponent allowed in the specified format (for example, 255_{10} for the single-precision format).

The signs of infinities are observed, and comparisons are possible. Infinities are always interpreted in the affine sense; that is, –∞ is less than any finite number and +∞ is greater than any finite number. Arithmetic on infinities is always exact. Exceptions are generated only when the use of an infinity as a source operand constitutes an invalid operation.

Whereas denormalized numbers may represent an underflow condition, the two ∞ numbers may represent the result of an overflow condition. Here, the normalized result of a computation has a biased exponent greater than the largest allowable exponent for the selected result format.

4.8.3.4 NaNs

Since NaNs are non-numbers, they are not part of the real number line. In Figure 4-12, the encoding space for NaNs in the floating-point formats is shown above the ends of the real number line. This space includes any value with the maximum allowable biased exponent and a non-zero fraction (the sign bit is ignored for NaNs).

The IA-32 architecture defines two classes of NaNs: quiet NaNs (QNaNs) and signaling NaNs (SNaNs). A QNaN is a NaN with the most significant fraction bit set; an SNaN is a NaN with the most significant fraction bit clear. QNaNs are allowed to propagate through most arithmetic operations without signaling an exception. SNaNs generally signal a floating-point invalid-operation exception whenever they appear as operands in arithmetic operations.

SNaNs are typically used to trap or invoke an exception handler. They must be inserted by software; that is, the processor never generates an SNaN as a result of a floating-point operation.

4.8.3.5 Operating on SNaNs and QNaNs

When a floating-point operation is performed on an SNaN and/or a QNaN, the result of the operation is either a QNaN delivered to the destination operand or the generation of a floating-point invalid operation exception, depending on the following rules:

- If one of the source operands is an SNaN and the floating-point invalid-operation exception is not masked (see Section 4.9.1.1, "Invalid Operation Exception (#I)"), then a floating-point invalid-operation exception is signaled and no result is stored in the destination operand. If one of the source operands is a QNaN and the floating-point invalid-operation exception is not masked and the operation is one that generates an invalid-operation exception for QNaN operands as described in Section 8.5.1.2, "Invalid Arithmetic Operand Exception (#IA)," or Section 11.5.2.1, "Invalid Operation Exception (#I)", then a floating-point invalid-operation exception is signaled and no result is stored in the destination operand.
- If either or both of the source operands are NaNs and floating-point invalid-operation exception is masked, the result is as shown in Table 4-7. When an SNaN is converted to a QNaN, the conversion is handled by setting the most-significant fraction bit of the SNaN to 1. Also, when one of the source operands is an SNaN, or when it is a QNaN and the operation is one that generates an invalid-operation exception for QNaN operands as described in Section 8.5.1.2, "Invalid Arithmetic Operand Exception (#IA)," or Section 11.5.2.1, "Invalid Operation Exception (#I)", then the floating-point invalid-operation exception flag is set. Note that for some combinations of source operands, the result is different for x87 FPU operations and for SSE/SSE2/SSE3/SSE4.1 operations. Intel AVX follows the same behavior as SSE/SSE2/SSE3/SSE4.1 in this respect.
- When neither of the source operands is a NaN, but the operation generates a floating-point invalid-operation exception (see Tables 8-10 and 11-1), the result is commonly a QNaN FP Indefinite (Section 4.8.3.7).

Any exceptions to the behavior described in Table 4-7 are described in Section 8.5.1.2, "Invalid Arithmetic Operand Exception (#IA)," and Section 11.5.2.1, "Invalid Operation Exception (#I)."

Table 4-7. Rules for Handling NaNs

Source Operands	Result[1]
SNaN and QNaN	x87 FPU — QNaN source operand. SSE/SSE2/SSE3/SSE4.1/AVX — First source operand (if this operand is an SNaN, it is converted to a QNaN)
Two SNaNs	x87 FPU—SNaN source operand with the larger significand, converted into a QNaN SSE/SSE2/SSE3/SSE4.1/AVX — First source operand converted to a QNaN
Two QNaNs	x87 FPU — QNaN source operand with the larger significand SSE/SSE2/SSE3/SSE4.1/AVX — First source operand
SNaN and a floating-point value	SNaN source operand, converted into a QNaN
QNaN and a floating-point value	QNaN source operand
SNaN (for instructions that take only one operand)	SNaN source operand, converted into a QNaN
QNaN (for instructions that take only one operand)	QNaN source operand

NOTE:

1. For SSE/SSE2/SSE3/SSE4.1 instructions, the first operand is generally a source operand that becomes the destination operand. For AVX instructions, the first source operand is usually the 2nd operand in a non-destructive source syntax. Within the **Result** column, the x87 FPU notation also applies to the FISTTP instruction in SSE3; the SSE3 notation applies to the SIMD floating-point instructions.

4.8.3.6 Using SNaNs and QNaNs in Applications

Except for the rules given at the beginning of Section 4.8.3.4, "NaNs," for encoding SNaNs and QNaNs, software is free to use the bits in the significand of a NaN for any purpose. Both SNaNs and QNaNs can be encoded to carry and store data, such as diagnostic information.

By unmasking the invalid operation exception, the programmer can use signaling NaNs to trap to the exception handler. The generality of this approach and the large number of NaN values that are available provide the sophisticated programmer with a tool that can be applied to a variety of special situations.

For example, a compiler can use signaling NaNs as references to uninitialized (real) array elements. The compiler can preinitialize each array element with a signaling NaN whose significand contains the index (relative position) of the element. Then, if an application program attempts to access an element that it has not initialized, it can use the NaN placed there by the compiler. If the invalid operation exception is unmasked, an interrupt will occur, and the exception handler will be invoked. The exception handler can determine which element has been accessed, since the operand address field of the exception pointer will point to the NaN, and the NaN will contain the index number of the array element.

Quiet NaNs are often used to speed up debugging. In its early testing phase, a program often contains multiple errors. An exception handler can be written to save diagnostic information in memory whenever it is invoked. After storing the diagnostic data, it can supply a quiet NaN as the result of the erroneous instruction, and that NaN can point to its associated diagnostic area in memory. The program will then continue, creating a different NaN for each error. When the program ends, the NaN results can be used to access the diagnostic data saved at the time the errors occurred. Many errors can thus be diagnosed and corrected in one test run.

In embedded applications that use computed results in further computations, an undetected QNaN can invalidate all subsequent results. Such applications should therefore periodically check for QNaNs and provide a recovery mechanism to be used if a QNaN result is detected.

4.8.3.7 QNaN Floating-Point Indefinite

For the floating-point data type encodings (single-precision, double-precision, and double-extended-precision), one unique encoding (a QNaN) is reserved for representing the special value QNaN floating-point indefinite. The x87 FPU and the SSE/SSE2/SSE3/SSE4.1/AVX extensions return these indefinite values as responses to some masked floating-point exceptions. Table 4-3 shows the encoding used for the QNaN floating-point indefinite.

4.8.3.8 Half-Precision Floating-Point Operation

Half-precision floating-point values are not used by the processor directly for arithmetic operations. Two instructions, VCVTPH2PS, VCVTPS2PH, provide conversion only between half-precision and single-precision floating-point values.

The SIMD floating-point exception behavior of VCVTPH2PS and VCVTPS2PH are described in Section 14.4.1.

4.8.4 Rounding

When performing floating-point operations, the processor produces an infinitely precise floating-point result in the destination format (single-precision, double-precision, or double extended-precision floating-point) whenever possible. However, because only a subset of the numbers in the real number continuum can be represented in IEEE Standard 754 floating-point formats, it is often the case that an infinitely precise result cannot be encoded exactly in the format of the destination operand.

For example, the following value (*a*) has a 24-bit fraction. The least-significant bit of this fraction (the underlined bit) cannot be encoded exactly in the single-precision format (which has only a 23-bit fraction):

(*a*) 1.0001 0000 1000 0011 1001 011$\underline{1}$E$_2$ 101

To round this result (*a*), the processor first selects two representable fractions *b* and *c* that most closely bracket *a* in value ($b < a < c$).

(*b*) 1.0001 0000 1000 0011 1001 011E$_2$ 101

(*c*) 1.0001 0000 1000 0011 1001 100E$_2$ 101

The processor then sets the result to *b* or to *c* according to the selected rounding mode. Rounding introduces an error in a result that is less than one unit in the last place (the least significant bit position of the floating-point value) to which the result is rounded.

The IEEE Standard 754 defines four rounding modes (see Table 4-8): round to nearest, round up, round down, and round toward zero. The default rounding mode (for the Intel 64 and IA-32 architectures) is round to nearest. This mode provides the most accurate and statistically unbiased estimate of the true result and is suitable for most applications.

Table 4-8. Rounding Modes and Encoding of Rounding Control (RC) Field

Rounding Mode	RC Field Setting	Description
Round to nearest (even)	00B	Rounded result is the closest to the infinitely precise result. If two values are equally close, the result is the even value (that is, the one with the least-significant bit of zero). Default
Round down (toward $-\infty$)	01B	Rounded result is closest to but no greater than the infinitely precise result.
Round up (toward $+\infty$)	10B	Rounded result is closest to but no less than the infinitely precise result.
Round toward zero (Truncate)	11B	Rounded result is closest to but no greater in absolute value than the infinitely precise result.

The round up and round down modes are termed **directed rounding** and can be used to implement interval arithmetic. Interval arithmetic is used to determine upper and lower bounds for the true result of a multistep computation, when the intermediate results of the computation are subject to rounding.

The round toward zero mode (sometimes called the "chop" mode) is commonly used when performing integer arithmetic with the x87 FPU.

The rounded result is called the inexact result. When the processor produces an inexact result, the floating-point precision (inexact) flag (PE) is set (see Section 4.9.1.6, "Inexact-Result (Precision) Exception (#P)").

The rounding modes have no effect on comparison operations, operations that produce exact results, or operations that produce NaN results.

4.8.4.1 Rounding Control (RC) Fields

In the Intel 64 and IA-32 architectures, the rounding mode is controlled by a 2-bit rounding-control (RC) field (Table 4-8 shows the encoding of this field). The RC field is implemented in two different locations:

- x87 FPU control register (bits 10 and 11)
- The MXCSR register (bits 13 and 14)

Although these two RC fields perform the same function, they control rounding for different execution environments within the processor. The RC field in the x87 FPU control register controls rounding for computations performed with the x87 FPU instructions; the RC field in the MXCSR register controls rounding for SIMD floating-point computations performed with the SSE/SSE2 instructions.

4.8.4.2 Truncation with SSE and SSE2 Conversion Instructions

The following SSE/SSE2 instructions automatically truncate the results of conversions from floating-point values to integers when the result it inexact: CVTTPD2DQ, CVTTPS2DQ, CVTTPD2PI, CVTTPS2PI, CVTTSD2SI, CVTTSS2SI. Here, truncation means the round toward zero mode described in Table 4-8.

4.9 OVERVIEW OF FLOATING-POINT EXCEPTIONS

The following section provides an overview of floating-point exceptions and their handling in the IA-32 architecture. For information specific to the x87 FPU and to the SSE/SSE2/SSE3/SSE4.1 extensions, refer to the following sections:

- Section 8.4, "x87 FPU Floating-Point Exception Handling"

- Section 11.5, "SSE, SSE2, and SSE3 Exceptions"

When operating on floating-point operands, the IA-32 architecture recognizes and detects six classes of exception conditions:

- Invalid operation (#I)
- Divide-by-zero (#Z)
- Denormalized operand (#D)
- Numeric overflow (#O)
- Numeric underflow (#U)
- Inexact result (precision) (#P)

The nomenclature of "#" symbol followed by one or two letters (for example, #P) is used in this manual to indicate exception conditions. It is merely a short-hand form and is not related to assembler mnemonics.

NOTE

All of the exceptions listed above except the denormal-operand exception (#D) are defined in IEEE Standard 754.

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 precision exceptions are post-computation exceptions.

Each of the six exception classes has a corresponding flag bit (IE, ZE, OE, UE, DE, or PE) and mask bit (IM, ZM, OM, UM, DM, or PM). When one or more floating-point exception conditions are detected, the processor sets the appropriate flag bits, then takes one of two possible courses of action, depending on the settings of the corresponding mask bits:

- Mask bit set. Handles the exception automatically, producing a predefined (and often times usable) result, while allowing program execution to continue undisturbed.
- Mask bit clear. Invokes a software exception handler to handle the exception.

The masked (default) responses to exceptions have been chosen to deliver a reasonable result for each exception condition and are generally satisfactory for most floating-point applications. By masking or unmasking specific floating-point exceptions, programmers can delegate responsibility for most exceptions to the processor and reserve the most severe exception conditions for software exception handlers.

Because the exception flags are "sticky," they provide a cumulative record of the exceptions that have occurred since they were last cleared. A programmer can thus mask all exceptions, run a calculation, and then inspect the exception flags to see if any exceptions were detected during the calculation.

In the IA-32 architecture, floating-point exception flag and mask bits are implemented in two different locations:

- x87 FPU status word and control word. The flag bits are located at bits 0 through 5 of the x87 FPU status word and the mask bits are located at bits 0 through 5 of the x87 FPU control word (see Figures 8-4 and 8-6).
- MXCSR register. The flag bits are located at bits 0 through 5 of the MXCSR register and the mask bits are located at bits 7 through 12 of the register (see Figure 10-3).

Although these two sets of flag and mask bits perform the same function, they report on and control exceptions for different execution environments within the processor. The flag and mask bits in the x87 FPU status and control words control exception reporting and masking for computations performed with the x87 FPU instructions; the companion bits in the MXCSR register control exception reporting and masking for SIMD floating-point computations performed with the SSE/SSE2/SSE3 instructions.

Note that when exceptions are masked, the processor may detect multiple exceptions in a single instruction, because it continues executing the instruction after performing its masked response. For example, the processor can detect a denormalized operand, perform its masked response to this exception, and then detect numeric underflow.

See Section 4.9.2, "Floating-Point Exception Priority," for a description of the rules for exception precedence when more than one floating-point exception condition is detected for an instruction.

4.9.1 Floating-Point Exception Conditions

The following sections describe the various conditions that cause a floating-point exception to be generated and the masked response of the processor when these conditions are detected. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A & 3B,* list the floating-point exceptions that can be signaled for each floating-point instruction.

4.9.1.1 Invalid Operation Exception (#I)

The processor reports an invalid operation exception in response to one or more invalid arithmetic operands. If the invalid operation exception is masked, the processor sets the IE flag and returns an indefinite value or a QNaN. This value overwrites the destination register specified by the instruction. If the invalid operation exception is not masked, the IE flag is set, a software exception handler is invoked, and the operands remain unaltered.

See Section 4.8.3.6, "Using SNaNs and QNaNs in Applications," for information about the result returned when an exception is caused by an SNaN.

The processor can detect a variety of invalid arithmetic operations that can be coded in a program. These operations generally indicate a programming error, such as dividing ∞ by ∞. See the following sections for information regarding the invalid-operation exception when detected while executing x87 FPU or SSE/SSE2/SSE3/SSE4.1 or AVX instructions:

- x87 FPU; Section 8.5.1, "Invalid Operation Exception".
- SIMD floating-point exceptions; Section 11.5.2.1, "Invalid Operation Exception (#I)".

4.9.1.2 Denormal Operand Exception (#D)

The processor reports the denormal-operand exception if an arithmetic instruction attempts to operate on a denormal operand (see Section 4.8.3.2, "Normalized and Denormalized Finite Numbers"). When the exception is masked, the processor sets the DE flag and proceeds with the instruction. Operating on denormal numbers will produce results at least as good as, and often better than, what can be obtained when denormal numbers are flushed to zero. Programmers can mask this exception so that a computation may proceed, then analyze any loss of accuracy when the final result is delivered.

When a denormal-operand exception is not masked, the DE flag is set, a software exception handler is invoked, and the operands remain unaltered. When denormal operands have reduced significance due to loss of low-order bits, it may be advisable to not operate on them. Precluding denormal operands from computations can be accomplished by an exception handler that responds to unmasked denormal-operand exceptions.

See the following sections for information regarding the denormal-operand exception when detected while executing x87 FPU or SSE/SSE2/SSE3 instructions:

- x87 FPU; Section 8.5.2, "Denormal Operand Exception (#D)".
- SIMD floating-point exceptions; Section 11.5.2.2, "Denormal-Operand Exception (#D)".

4.9.1.3 Divide-By-Zero Exception (#Z)

The processor reports the floating-point divide-by-zero exception whenever an instruction attempts to divide a finite non-zero operand by 0. The masked response for the divide-by-zero exception is to set the ZE flag and return an infinity signed with the exclusive OR of the sign of the operands. If the divide-by-zero exception is not masked, the ZE flag is set, a software exception handler is invoked, and the operands remain unaltered.

See the following sections for information regarding the divide-by-zero exception when detected while executing x87 FPU or SSE/SSE2 instructions:

- x87 FPU; Section 8.5.3, "Divide-By-Zero Exception (#Z)".
- SIMD floating-point exceptions; Section 11.5.2.3, "Divide-By-Zero Exception (#Z)".

4.9.1.4 Numeric Overflow Exception (#O)

The processor reports a floating-point numeric overflow exception whenever the rounded result of an instruction exceeds the largest allowable finite value that will fit into the destination operand. Table 4-9 shows the threshold range for numeric overflow for each of the floating-point formats; overflow occurs when a rounded result falls at or outside this threshold range.

Table 4-9. Numeric Overflow Thresholds

Floating-Point Format	Overflow Thresholds
Single Precision	$\|x\| \geq 1.0 * 2^{128}$
Double Precision	$\|x\| \geq 1.0 * 2^{1024}$
Double Extended Precision	$\|x\| \geq 1.0 * 2^{16384}$

When a numeric-overflow exception occurs and the exception is masked, the processor sets the OE flag and returns one of the values shown in Table 4-10, according to the current rounding mode. See Section 4.8.4, "Rounding."

When numeric overflow occurs and the numeric-overflow exception is not masked, the OE flag is set, a software exception handler is invoked, and the source and destination operands either remain unchanged or a biased result is stored in the destination operand (depending whether the overflow exception was generated during an SSE/SSE2/SSE3 floating-point operation or an x87 FPU operation).

Table 4-10. Masked Responses to Numeric Overflow

Rounding Mode	Sign of True Result	Result
To nearest	+	+∞
	–	–∞
Toward –∞	+	Largest finite positive number
	–	–∞
Toward +∞	+	+∞
	–	Largest finite negative number
Toward zero	+	Largest finite positive number
	–	Largest finite negative number

See the following sections for information regarding the numeric overflow exception when detected while executing x87 FPU instructions or while executing SSE/SSE2/SSE3 instructions:

- x87 FPU; Section 8.5.4, "Numeric Overflow Exception (#O)"
- SIMD floating-point exceptions; Section 11.5.2.4, "Numeric Overflow Exception (#O)"

4.9.1.5 Numeric Underflow Exception (#U)

The processor detects a potential floating-point numeric underflow condition whenever the result of rounding with unbounded exponent (taking into account precision control for x87) is non-zero and tiny; that is, non-zero and less than the smallest possible normalized, finite value that will fit into the destination operand. Table 4-11 shows the threshold range for numeric underflow for each of the floating-point formats (assuming normalized results); underflow occurs when a rounded result falls strictly within the threshold range. The ability to detect and handle underflow is provided to prevent a very small result from propagating through a computation and causing another exception (such as overflow during division) to be generated at a later time. Results which trigger underflow are also potentially less accurate.

Table 4-11. Numeric Underflow (Normalized) Thresholds

Floating-Point Format	Underflow Thresholds*
Single Precision	$\|x\| < 1.0 * 2^{-126}$
Double Precision	$\|x\| < 1.0 * 2^{-1022}$
Double Extended Precision	$\|x\| < 1.0 * 2^{-16382}$

* Where 'x' is the result rounded to destination precision with an unbounded exponent range.

How the processor handles an underflow condition, depends on two related conditions:

- creation of a tiny, non-zero result
- creation of an inexact result; that is, a result that cannot be represented exactly in the destination format

Which of these events causes an underflow exception to be reported and how the processor responds to the exception condition depends on whether the underflow exception is masked:

- **Underflow exception masked —** The underflow exception is reported (the UE flag is set) only when the result is both tiny and inexact. The processor returns a correctly signed result whose magnitude is less than or equal to the smallest positive normal floating-point number to the destination operand, regardless of inexactness.
- **Underflow exception not masked —** The underflow exception is reported when the result is non-zero tiny, regardless of inexactness. The processor leaves the source and destination operands unaltered or stores a biased result in the destination operand (depending whether the underflow exception was generated during an SSE/SSE2/SSE3 floating-point operation or an x87 FPU operation) and invokes a software exception handler.

See the following sections for information regarding the numeric underflow exception when detected while executing x87 FPU instructions or while executing SSE/SSE2/SSE3 instructions:

- x87 FPU; Section 8.5.5, "Numeric Underflow Exception (#U)"
- SIMD floating-point exceptions; Section 11.5.2.5, "Numeric Underflow Exception (#U)"

4.9.1.6 Inexact-Result (Precision) Exception (#P)

The inexact-result exception (also called the precision exception) occurs if the result of an operation is not exactly representable in the destination format. For example, the fraction 1/3 cannot be precisely represented in binary floating-point form. This exception occurs frequently and indicates that some (normally acceptable) accuracy will be lost due to rounding. The exception is supported for applications that need to perform exact arithmetic only. Because the rounded result is generally satisfactory for most applications, this exception is commonly masked.

If the inexact-result exception is masked when an inexact-result condition occurs and a numeric overflow or underflow condition has not occurred, the processor sets the PE flag and stores the rounded result in the destination operand. The current rounding mode determines the method used to round the result. See Section 4.8.4, "Rounding."

If the inexact-result exception is not masked when an inexact result occurs and numeric overflow or underflow has not occurred, the PE flag is set, the rounded result is stored in the destination operand, and a software exception handler is invoked.

If an inexact result occurs in conjunction with numeric overflow or underflow, one of the following operations is carried out:

- If an inexact result occurs along with masked overflow or underflow, the OE flag or UE flag and the PE flag are set and the result is stored as described for the overflow or underflow exceptions; see Section 4.9.1.4, "Numeric Overflow Exception (#O)," or Section 4.9.1.5, "Numeric Underflow Exception (#U)." If the inexact result exception is unmasked, the processor also invokes a software exception handler.
- If an inexact result occurs along with unmasked overflow or underflow and the destination operand is a register, the OE or UE flag and the PE flag are set, the result is stored as described for the overflow or underflow exceptions, and a software exception handler is invoked.

If an unmasked numeric overflow or underflow exception occurs and the destination operand is a memory location (which can happen only for a floating-point store), the inexact-result condition is not reported and the C1 flag is cleared.

See the following sections for information regarding the inexact-result exception when detected while executing x87 FPU or SSE/SSE2/SSE3 instructions:

- x87 FPU; Section 8.5.6, "Inexact-Result (Precision) Exception (#P)"
- SIMD floating-point exceptions; Section 11.5.2.3, "Divide-By-Zero Exception (#Z)"

4.9.2 Floating-Point Exception Priority

The processor handles exceptions according to a predetermined precedence. When an instruction generates two or more exception conditions, the exception precedence sometimes results in the higher-priority exception being handled and the lower-priority exceptions being ignored. For example, dividing an SNaN by zero can potentially signal an invalid-operation exception (due to the SNaN operand) and a divide-by-zero exception. Here, if both exceptions are masked, the processor handles the higher-priority exception only (the invalid-operation exception), returning a QNaN to the destination. Alternately, a denormal-operand or inexact-result exception can accompany a numeric underflow or overflow exception with both exceptions being handled.

The precedence for floating-point exceptions is as follows:

1. Invalid-operation exception, subdivided as follows:
   a. stack underflow (occurs with x87 FPU only)
   b. stack overflow (occurs with x87 FPU only)
   c. operand of unsupported format (occurs with x87 FPU only when using the double extended-precision floating-point format)
   d. SNaN operand
2. QNaN operand. Though 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 zero-divide exception.
3. Any other invalid-operation exception not mentioned above or a divide-by-zero exception.
4. Denormal-operand exception. If masked, then instruction execution continues and a lower-priority exception can occur as well.
5. Numeric overflow and underflow exceptions; possibly in conjunction with the inexact-result exception.
6. Inexact-result exception.

Invalid operation, zero divide, and denormal operand exceptions are detected before a floating-point operation begins. Overflow, underflow, and precision exceptions are not detected until a true result has been computed. When an unmasked **pre-operation** exception is detected, the destination operand has not yet been updated, and appears as if the offending instruction has not been executed. When an unmasked **post-operation** exception is detected, the destination operand may be updated with a result, depending on the nature of the exception (except for SSE/SSE2/SSE3 instructions, which do not update their destination operands in such cases).

4.9.3 Typical Actions of a Floating-Point Exception Handler

After the floating-point exception handler is invoked, the processor handles the exception in the same manner that it handles non-floating-point exceptions. The floating-point exception handler is normally part of the operating system or executive software, and it usually invokes a user-registered floating-point exception handle.

A typical action of the exception handler is to store state information in memory. Other typical exception handler actions include:

- Examining the stored state information to determine the nature of the error
- Taking actions to correct the condition that caused the error
- Clearing the exception flags
- Returning to the interrupted program and resuming normal execution

In lieu of writing recovery procedures, the exception handler can do the following:

- Increment in software an exception counter for later display or printing

- Print or display diagnostic information (such as the state information)
- Halt further program execution

3. Updates to Chapter 5, Volume 1

Change bars and green 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: Added the 12th generation Intel® Core™ processor to applicable rows in Table 5-2, "Instruction Set Extensions Introduction in Intel 64 and IA-32 Processors ".

CHAPTER 5
INSTRUCTION SET SUMMARY

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, “SSE Instructions”.
- Section 5.6, “SSE2 Instructions”.
- Section 5.7, “SSE3 Instructions”.
- Section 5.8, “Supplemental Streaming SIMD Extensions 3 (SSSE3) Instructions”.
- Section 5.9, “SSE4 Instructions”.
- Section 5.10, “SSE4.1 Instructions”.
- Section 5.11, “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”.

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.

Table 5-1. Instruction Groups in Intel 64 and IA-32 Processors (Contd.)

Instruction Set Architecture	Intel 64 and IA-32 Processor Support
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 90nm process technology), Intel Core Solo, Intel Core Duo, Intel Core 2 Duo processors, Intel Xeon processor 3xxxx, 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.
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.

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

Instruction Set Architecture	Processor Generation Introduction
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® Processor Scalable Family, Intel® Core™ i3-8121U processor.
CLFLUSHOPT, XSAVEC, XSAVES, Intel® MPX	Intel Xeon Processor Scalable 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 Processor Scalable Family, Intel Core i3-8121U processor based on Cannon Lake microarchitecture.
CLWB	Intel Xeon Processor Scalable Family, Intel Atom® processor based on Tremont microarchitecture, 11th Generation Intel Core processor family based on Tiger Lake microarchitecture.
PKU	Intel Xeon Processor Scalable 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® Processor Scalable 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 based on Alder Lake performance hybrid architecture.
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® Processor Scalable 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 MOV	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.
VAES, GFNI (AVX/AVX512), AVX512_VBMI2, VPCLMULQDQ, AVX512_BITALG	10th Generation Intel Core processor family based on Ice Lake microarchitecture.
ENCLV	Intel Atom processor based on Tremont microarchitecture, 3rd Generation Intel® Xeon® Processor Scalable Processors 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.
CLDEMOTE	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
Direct stores: MOVDIRI, MOVDIR64B	Intel Atom processor based on Tremont microarchitecture, 11th Generation Intel Core processor family based on Tiger Lake 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.
AVX512_BF16	3rd Generation Intel® Xeon® Processor Scalable Processors based on Cooper Lake product.
AVX512_VP2INTERSECT	11th Generation Intel Core processor family based on Tiger Lake microarchitecture.
Key Locker[1]	11th Generation Intel Core processor family based on Tiger Lake microarchitecture.
Control-flow Enforcement Technology (CET)	11th Generation Intel Core processor family based on Tiger Lake microarchitecture.
MKTME[2], PCONFIG	3rd Generation Intel® Xeon® Processor Scalable Family based on Ice Lake microarchitecture.
WBNOINVD	3rd Generation Intel® Xeon® Processor Scalable Family based on Ice Lake microarchitecture.
LBRs (architectural)	12th generation Intel Core processor based on Alder Lake performance hybrid architecture.
Intel® Virtualization Technology - Redirect Protection (Intel® VT-rp) and HLAT	12th generation Intel Core processor based on Alder Lake performance hybrid architecture.
AVX-VNNI	12th generation Intel Core processor based on Alder Lake performance hybrid architecture[3].
SERIALIZE	12th generation Intel Core processor based on Alder Lake performance hybrid architecture.
Intel® Thread Director and HRESET	12th generation Intel Core processor based on Alder Lake performance hybrid architecture.
Fast zero-length REP MOVSB, fast short REP STOSB	12th generation Intel Core processor based on Alder Lake performance hybrid architecture.
Fast Short REP CMPSB, fast short REP SCASB	Future Intel Xeon processors.
Supervisor Memory Protection Keys (PKS)	12th generation Intel Core processor based on Alder Lake performance hybrid architecture, future Intel processors.

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 MKTME 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.
CMOVB/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]	This instruction 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.

1. Processor support of CRC32 is enumerated by CPUID.01:ECX[SSE4.2] = 1
2. Processor support of POPCNT is enumerated by CPUID.01:ECX[POPCNT] = 1

JA/JNBE	Jump if above/Jump if not below or equal.
JAE/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.
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/MOVSB	Move string/Move byte string.
MOVS/MOVSW	Move string/Move word string.
MOVS/MOVSD	Move string/Move doubleword string.
CMPS/CMPSB	Compare string/Compare byte string.
CMPS/CMPSW	Compare string/Compare word string.
CMPS/CMPSD	Compare string/Compare doubleword string.
SCAS/SCASB	Scan string/Scan byte string.
SCAS/SCASW	Scan string/Scan word string.
SCAS/SCASD	Scan string/Scan doubleword string.
LODS/LODSB	Load string/Load byte string.
LODS/LODSW	Load string/Load word string.
LODS/LODSD	Load string/Load doubleword string.
STOS/STOSB	Store string/Store byte string.
STOS/STOSW	Store string/Store word string.

STOS/STOSD	Store string/Store doubleword string.
REP	Repeat while ECX not zero.
REPE/REPZ	Repeat while equal/Repeat while zero.
REPNE/REPNZ	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/POPFD	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.
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 Sate 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, BMI2

ANDN	Bitwise AND of first source with inverted 2nd 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 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 trailing zero bits.

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

5.1.16.1 Detection of VEX-encoded GPR Instructions, LZCNT and TZCNT, 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.(EAX=07H, ECX=0H):EBX.BMI1[bit 3]: if 1 indicates the processor supports the first group of advanced bit manipulation extensions (ANDN, BEXTR, BLSI, BLSMSK, BLSR, TZCNT);

CPUID.(EAX=07H, ECX=0H):EBX.BMI2[bit 8]: if 1 indicates the processor supports the second group of advanced bit manipulation extensions (BZHI, MULX, PDEP, PEXT, RORX, SARX, SHLX, SHRX);

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

CPUID.EAX=80000001H:ECX.PREFTEHCHW[bit 8]: if 1 indicates the processor supports the PREFTEHCHW instruction. CPUID.(EAX=07H, ECX=0H):ECX.PREFTEHCHWT1[bit 0]: if 1 indicates the processor supports the PREFTEHCHWT1 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.

1. SSE3 provides an instruction FISTTP for integer conversion.

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
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.
PADDD	Add packed doubleword integers.
PADDSB	Add packed signed byte integers with signed saturation.
PADDSW	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 SSE INSTRUCTIONS

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)."

SSE instructions can only be executed on Intel 64 and IA-32 processors that support 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.*

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 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 SSE Data Transfer Instructions

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 and XMM register and memory.
MOVUPS	Move four unaligned packed single-precision floating-point values between XMM registers or between and XMM register and memory.
MOVHPS	Move two packed single-precision floating-point values to an 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 an 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 SSE Packed Arithmetic Instructions

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 SSE Comparison Instructions

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 SSE Logical Instructions

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 SSE Shuffle and Unpack Instructions

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 SSE Conversion Instructions

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 doubleword integer to scalar single-precision floating-point value.
CVTPS2PI	Convert packed single-precision floating-point values to packed doubleword integers.
CVTTPS2PI	Convert with truncation packed single-precision floating-point values to packed doubleword integers.
CVTSS2SI	Convert a scalar single-precision floating-point value to a doubleword integer.
CVTTSS2SI	Convert with truncation a scalar single-precision floating-point value to a scalar doubleword integer.

5.5.2 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 SSE 64-Bit SIMD Integer Instructions

These 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.
PMOVMSKB	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 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 PREFETCH*h* 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.
PREFETCH*h*	Load 32 or more of bytes from memory to a selected level of the processor's cache hierarchy
SFENCE	Serializes store operations.

5.6 SSE2 INSTRUCTIONS

SSE2 extensions represent an extension of the SIMD execution model introduced with MMX technology and the SSE extensions. 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)."

SSE2 instructions can only be executed on Intel 64 and IA-32 processors that support the 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 SSE2 Packed and Scalar Double-Precision Floating-Point Instructions

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 SSE2 Data Movement Instructions

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 and XMM register and memory.
MOVUPD	Move two unaligned packed double-precision floating-point values between XMM registers or between and XMM register and memory.
MOVHPD	Move high packed double-precision floating-point value to an from the high quadword of an XMM register and memory.
MOVLPD	Move low packed single-precision floating-point value to an 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 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 SSE2 Logical Instructions

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 SSE2 Compare Instructions

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 SSE2 Shuffle and Unpack Instructions

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 SSE2 Conversion Instructions

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 doubleword integer.
CVTTSD2SI	Convert with truncation scalar double-precision floating-point values to scalar doubleword integers.
CVTSI2SD	Convert doubleword integer to scalar double-precision floating-point value.

5.6.2 SSE2 Packed Single-Precision Floating-Point Instructions

SSE2 packed single-precision floating-point instructions perform conversion operations on single-precision floating-point and integer operands. These instructions represent enhancements to the 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 SSE2 128-Bit SIMD Integer Instructions

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.
PSHUFD	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 SSE2 Cacheability Control and Ordering Instructions

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 SSE3 INSTRUCTIONS

The 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 x87FPU 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.

SSE3 instructions can only be executed on Intel 64 and IA-32 processors that support 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 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 SSE3 Specialized 128-bit Unaligned Data Load Instruction

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

5.7.3 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 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 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 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.
PHADDD	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

PMADDUBSW	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

PMULHRSW	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

PSHUFB — 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 SSE4 INSTRUCTIONS

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

SSE4.1 is targeted to improve the performance of media, imaging, and 3D workloads. 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 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 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 SSE4.1 INSTRUCTIONS

SSE4.1 instructions can use an XMM register as a source or destination. Programming SSE4.1 is similar to programming 128-bit Integer SIMD and floating-point SIMD instructions in SSE/SSE2/SSE3/SSSE3. 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.
PMINSD	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.
PMOVZXBD	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	PACKUSDW packs dword to word with unsigned saturation.

5.11 SSE4.2 INSTRUCTION SET

Five of the 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 SSE4.2 instructions is similar to programming 128-bit Integer SIMD in SSE2/SSSE3. 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) promotes legacy 128-bit SIMD instruction sets that operate on 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 operates on XMM (but not on MMX registers) are promoted to support three-operand syntax with VEX-128 encoding.

VEX-prefix encoded 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 AVX instructions.

The list of AVX instructions are listed 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 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 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

Conversion between single-precision floating-point (32-bit) and half-precision FP (16-bit) data are provided by VCVTPS2PH, VCVTPH2PS:

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

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.

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.
SHA1RNDS4	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.
SHA256RNDS2	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 enhance-

ment provided by opmask registers not available to VEX-encoded Intel AVX / Intel AVX2. Some instruction mnemonics in AVX / AVX2 that are promoted into 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 2.6, "Intel® AVX-512 Encoding" of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

512-bit instruction mnemonics in AVX-512F that are not AVX/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 AVX/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.
VRANGEPD/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.
VREDUCEPD/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 AVX/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.
VPERMI2B/W	Full permute from two tables of byte/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 AVX/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.

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.
INVPCID	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 systemcall.

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	This instruction 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[SENTER]	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[SMCRTL]	SMX mode control.
GETSEC[WAKEUP]	Wakes up sleeping logical processors inside an MLE.

5.24 INTEL® MEMORY PROTECTION EXTENSIONS

Intel Memory Protection Extensions (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 Chapter 17, "Intel® MPX".

BNDMK	Create a LowerBound and a 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 compliment form.
BNDCN	Check the address of a memory reference against an UpperBound not in 1's compliment 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 33 through CHAPTER 39 of *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 is 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[EDBGRD]	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		
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.

4. Updates to Chapter 8, Volume 1

Change bars and green text show changes to Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

Changes to this chapter: Update to Section 8.2.2, "Unsupported Double Extended-Precision Floating-Point Encodings and Pseudo-Denormals".

CHAPTER 8
PROGRAMMING WITH THE X87 FPU

The x87 Floating-Point Unit (FPU) provides high-performance floating-point processing capabilities for use in graphics processing, scientific, engineering, and business applications. It supports the floating-point, integer, and packed BCD integer data types and the floating-point processing algorithms and exception handling architecture defined in the IEEE Standard 754 for Binary Floating-Point Arithmetic.

This chapter describes the x87 FPU's execution environment and instruction set. It also provides exception handling information that is specific to the x87 FPU. Refer to the following chapters or sections of chapters for additional information about x87 FPU instructions and floating-point operations:

- *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A & 2B,* provide detailed descriptions of x87 FPU instructions.
- Section 4.2.2, "Floating-Point Data Types," Section 4.2.1.2, "Signed Integers," and Section 4.7, "BCD and Packed BCD Integers," describe the floating-point, integer, and BCD data types.
- Section 4.9, "Overview of Floating-Point Exceptions," Section 4.9.1, "Floating-Point Exception Conditions," and Section 4.9.2, "Floating-Point Exception Priority," give an overview of the floating-point exceptions that the x87 FPU can detect and report.

8.1 X87 FPU EXECUTION ENVIRONMENT

The x87 FPU represents a separate execution environment within the IA-32 architecture (see Figure 8-1). This execution environment consists of eight data registers (called the x87 FPU data registers) and the following special-purpose registers:

- Status register
- Control register
- Tag word register
- Last instruction pointer register
- Last data (operand) pointer register
- Opcode register

These registers are described in the following sections.

The x87 FPU executes instructions from the processor's normal instruction stream. The state of the x87 FPU is independent from the state of the basic execution environment and from the state of SSE/SSE2/SSE3 extensions.

However, the x87 FPU and Intel MMX technology share state because the MMX registers are aliased to the x87 FPU data registers. Therefore, when writing code that uses x87 FPU and MMX instructions, the programmer must explicitly manage the x87 FPU and MMX state (see Section 9.5, "Compatibility with x87 FPU Architecture").

8.1.1 x87 FPU in 64-Bit Mode and Compatibility Mode

In compatibility mode and 64-bit mode, x87 FPU instructions function like they do in protected mode. Memory operands are specified using the ModR/M, SIB encoding that is described in Section 3.7.5, "Specifying an Offset."

8.1.2 x87 FPU Data Registers

The x87 FPU data registers (shown in Figure 8-1) consist of eight 80-bit registers. Values are stored in these registers in the double extended-precision floating-point format shown in Figure 4-3. When floating-point, integer, or packed BCD integer values are loaded from memory into any of the x87 FPU data registers, the values are automatically converted into double extended-precision floating-point format (if they are not already in that format). When computation results are subsequently transferred back into memory from any of the x87 FPU registers, the

results can be left in the double extended-precision floating-point format or converted back into a shorter floating-point format, an integer format, or the packed BCD integer format. (See Section 8.2, "x87 FPU Data Types," for a description of the data types operated on by the x87 FPU.)

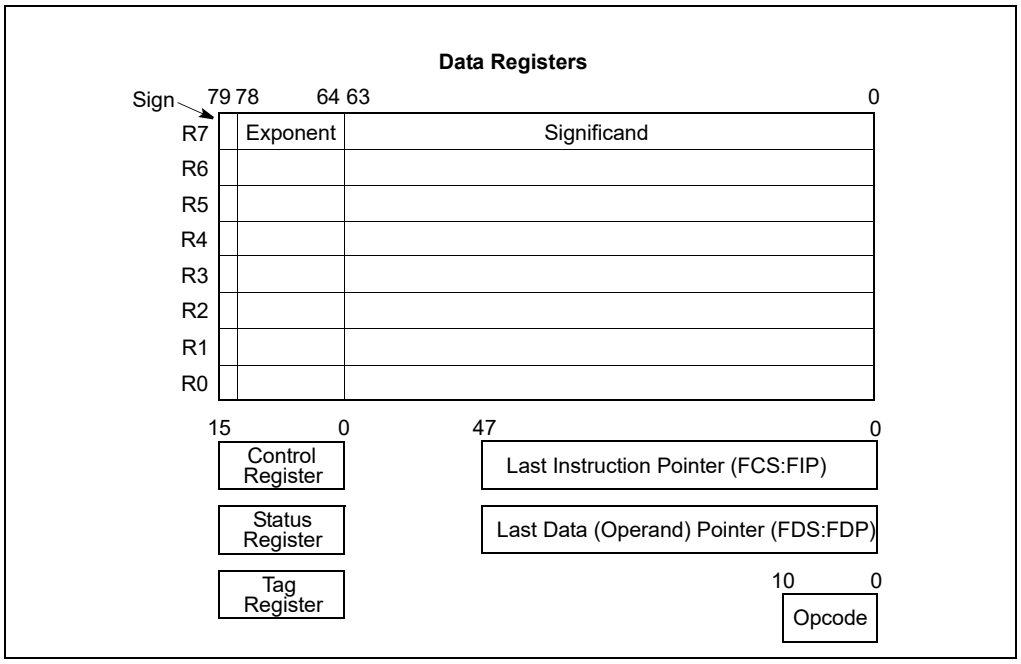


Figure 8-1. x87 FPU Execution Environment

The x87 FPU instructions treat the eight x87 FPU data registers as a register stack (see Figure 8-2). All addressing of the data registers is relative to the register on the top of the stack. The register number of the current top-of-stack register is stored in the TOP (stack TOP) field in the x87 FPU status word. Load operations decrement TOP by one and load a value into the new top-of-stack register, and store operations store the value from the current TOP register in memory and then increment TOP by one. (For the x87 FPU, a load operation is equivalent to a push and a store operation is equivalent to a pop.) Note that load and store operations are also available that do not push and pop the stack.

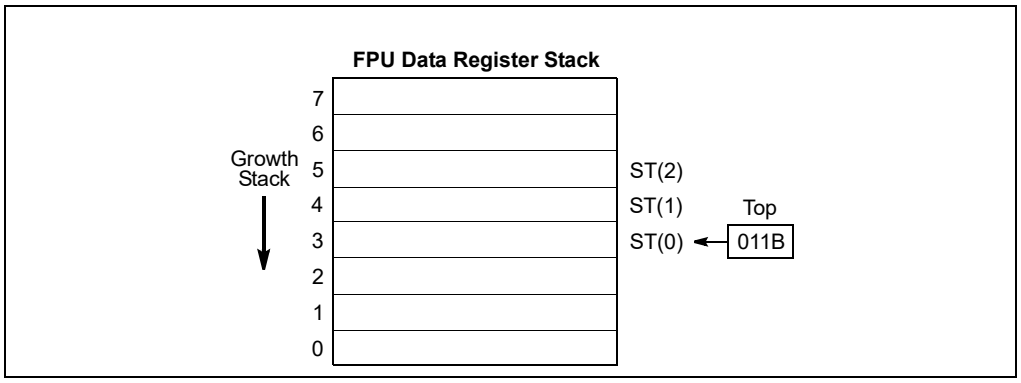


Figure 8-2. x87 FPU Data Register Stack

If a load operation is performed when TOP is at 0, register wraparound occurs and the new value of TOP is set to 7. The floating-point stack-overflow exception indicates when wraparound might cause an unsaved value to be overwritten (see Section 8.5.1.1, "Stack Overflow or Underflow Exception (#IS)").

Many floating-point instructions have several addressing modes that permit the programmer to implicitly operate on the top of the stack, or to explicitly operate on specific registers relative to the TOP. Assemblers support these

register addressing modes, using the expression ST(0), or simply ST, to represent the current stack top and ST(i) to specify the *i*th register from TOP in the stack ($0 \le i \le 7$). For example, if TOP contains 011B (register 3 is the top of the stack), the following instruction would add the contents of two registers in the stack (registers 3 and 5):

```
FADD ST, ST(2);
```

Figure 8-3 shows an example of how the stack structure of the x87 FPU registers and instructions are typically used to perform a series of computations. Here, a two-dimensional dot product is computed, as follows:

1. The first instruction (FLD value1) decrements the stack register pointer (TOP) and loads the value 5.6 from memory into ST(0). The result of this operation is shown in snap-shot (a).
2. The second instruction multiplies the value in ST(0) by the value 2.4 from memory and stores the result in ST(0), shown in snap-shot (b).
3. The third instruction decrements TOP and loads the value 3.8 in ST(0).
4. The fourth instruction multiplies the value in ST(0) by the value 10.3 from memory and stores the result in ST(0), shown in snap-shot (c).
5. The fifth instruction adds the value and the value in ST(1) and stores the result in ST(0), shown in snap-shot (d).

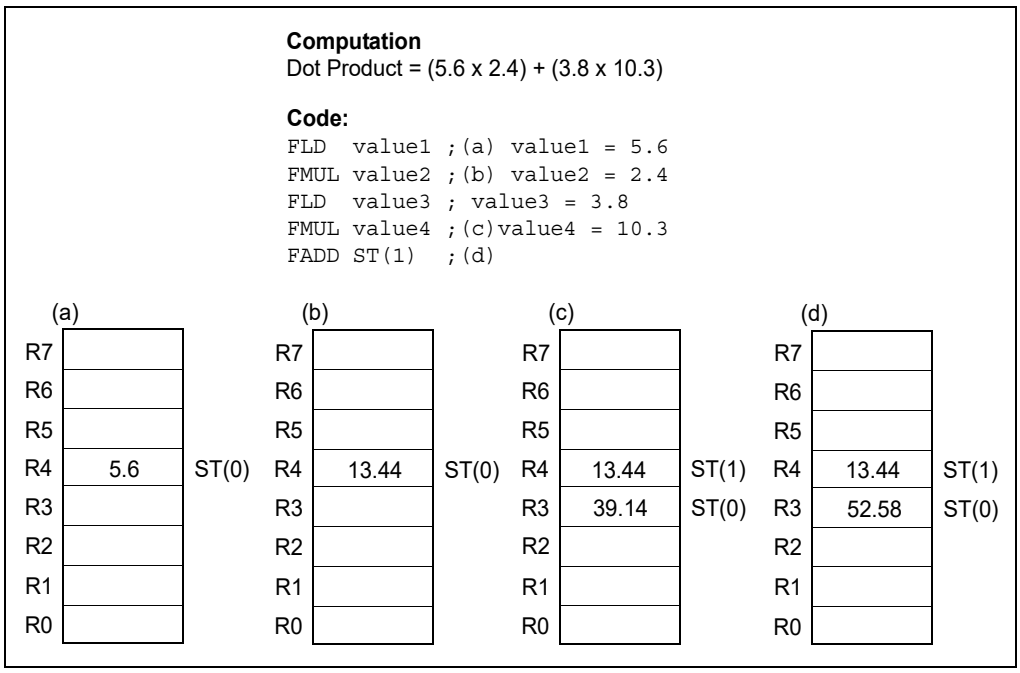


Figure 8-3. Example x87 FPU Dot Product Computation

The style of programming demonstrated in this example is supported by the floating-point instruction set. In cases where the stack structure causes computation bottlenecks, the FXCH (exchange x87 FPU register contents) instruction can be used to streamline a computation.

8.1.2.1 Parameter Passing With the x87 FPU Register Stack

Like the general-purpose registers, the contents of the x87 FPU data registers are unaffected by procedure calls, or in other words, the values are maintained across procedure boundaries. A calling procedure can thus use the x87 FPU data registers (as well as the procedure stack) for passing parameter between procedures. The called procedure can reference parameters passed through the register stack using the current stack register pointer (TOP) and the ST(0) and ST(i) nomenclature. It is also common practice for a called procedure to leave a return value or result in register ST(0) when returning execution to the calling procedure or program.

When mixing MMX and x87 FPU instructions in the procedures or code sequences, the programmer is responsible for maintaining the integrity of parameters being passed in the x87 FPU data registers. If an MMX instruction is executed before the parameters in the x87 FPU data registers have been passed to another procedure, the parameters may be lost (see Section 9.5, "Compatibility with x87 FPU Architecture").

8.1.3 x87 FPU Status Register

The 16-bit x87 FPU status register (see Figure 8-4) indicates the current state of the x87 FPU. The flags in the x87 FPU status register include the FPU busy flag, top-of-stack (TOP) pointer, condition code flags, exception summary status flag, stack fault flag, and exception flags. The x87 FPU sets the flags in this register to show the results of operations.

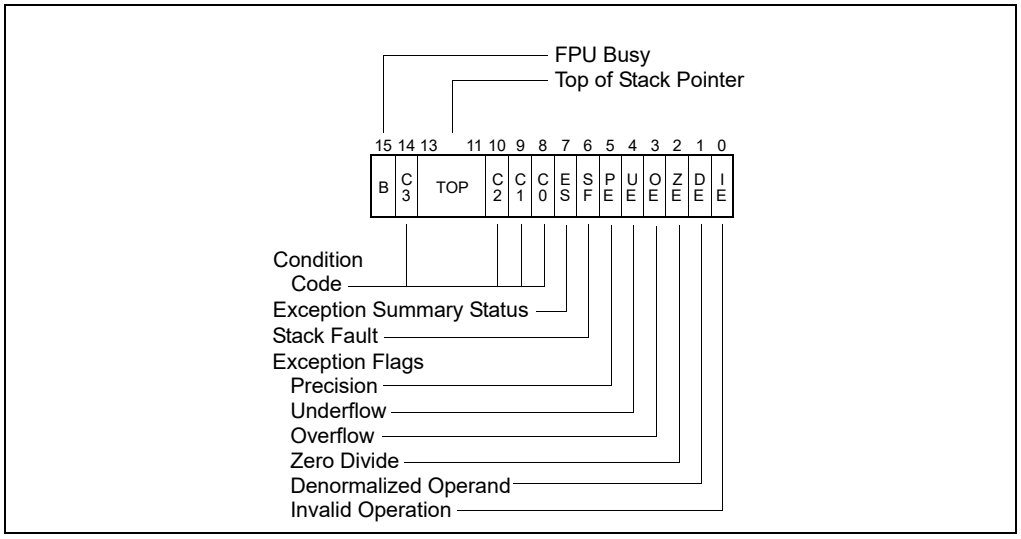


Figure 8-4. x87 FPU Status Word

The contents of the x87 FPU status register (referred to as the x87 FPU status word) can be stored in memory using the FSTSW/FNSTSW, FSTENV/FNSTENV, FSAVE/FNSAVE, and FXSAVE instructions. It can also be stored in the AX register of the integer unit, using the FSTSW/FNSTSW instructions.

8.1.3.1 Top of Stack (TOP) Pointer

A pointer to the x87 FPU data register that is currently at the top of the x87 FPU register stack is contained in bits 11 through 13 of the x87 FPU status word. This pointer, which is commonly referred to as TOP (for top-of-stack), is a binary value from 0 to 7. See Section 8.1.2, "x87 FPU Data Registers," for more information about the TOP pointer.

8.1.3.2 Condition Code Flags

The four condition code flags (C0 through C3) indicate the results of floating-point comparison and arithmetic operations. Table 8-1 summarizes the manner in which the floating-point instructions set the condition code flags. These condition code bits are used principally for conditional branching and for storage of information used in exception handling (see Section 8.1.4, "Branching and Conditional Moves on Condition Codes").

As shown in Table 8-1, the C1 condition code flag is used for a variety of functions. When both the IE and SF flags in the x87 FPU status word are set, indicating a stack overflow or underflow exception (#IS), the C1 flag distinguishes between overflow (C1 = 1) and underflow (C1 = 0). When the PE flag in the status word is set, indicating an inexact (rounded) result, the C1 flag is set to 1 if the last rounding by the instruction was upward. The FXAM instruction sets C1 to the sign of the value being examined.

The C2 condition code flag is used by the FPREM and FPREM1 instructions to indicate an incomplete reduction (or partial remainder). When a successful reduction has been completed, the C0, C3, and C1 condition code flags are set to the three least-significant bits of the quotient (Q2, Q1, and Q0, respectively). See "FPREM1—Partial Remainder" in Chapter 3, "Instruction Set Reference, A-L," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A,* for more information on how these instructions use the condition code flags.

The FPTAN, FSIN, FCOS, and FSINCOS instructions set the C2 flag to 1 to indicate that the source operand is beyond the allowable range of $\pm 2^{63}$ and clear the C2 flag if the source operand is within the allowable range.

Where the state of the condition code flags are listed as undefined in Table 8-1, do not rely on any specific value in these flags.

8.1.3.3 x87 FPU Floating-Point Exception Flags

The six x87 FPU floating-point exception flags (bits 0 through 5) of the x87 FPU status word indicate that one or more floating-point exceptions have been detected since the bits were last cleared. The individual exception flags (IE, DE, ZE, OE, UE, and PE) are described in detail in Section 8.4, "x87 FPU Floating-Point Exception Handling." Each of the exception flags can be masked by an exception mask bit in the x87 FPU control word (see Section 8.1.5, "x87 FPU Control Word"). The exception summary status flag (ES, bit 7) is set when any of the unmasked exception flags are set. When the ES flag is set, the x87 FPU exception handler is invoked, using one of the techniques described in Section 8.7, "Handling x87 FPU Exceptions in Software." (Note that if an exception flag is masked, the x87 FPU will still set the appropriate flag if the associated exception occurs, but it will not set the ES flag.)

The exception flags are "sticky" bits (once set, they remain set until explicitly cleared). They can be cleared by executing the FCLEX/FNCLEX (clear exceptions) instructions, by reinitializing the x87 FPU with the FINIT/FNINIT or FSAVE/FNSAVE instructions, or by overwriting the flags with an FRSTOR or FLDENV instruction.

The B-bit (bit 15) is included for 8087 compatibility only. It reflects the contents of the ES flag.

Table 8-1. Condition Code Interpretation

Instruction	C0	C3	C2	C1
FCOM, FCOMP, FCOMPP, FICOM, FICOMP, FTST, FUCOM, FUCOMP, FUCOMPP	Result of Comparison		Operands are not Comparable	0 or #IS
FCOMI, FCOMIP, FUCOMI, FUCOMIP	Undefined. (These instructions set the status flags in the EFLAGS register.)			#IS
FXAM	Operand class			Sign
FPREM, FPREM1	Q2	Q1	0 = reduction complete 1 = reduction incomplete	Q0 or #IS
F2XM1, FADD, FADDP, FBSTP, FCMOVcc, FIADD, FDIV, FDIVP, FDIVR, FDIVRP, FIDIV, FIDIVR, FIMUL, FIST, FISTP, FISUB, FISUBR,FMUL, FMULP, FPATAN, FRNDINT, FSCALE, FST, FSTP, FSUB, FSUBP, FSUBR, FSUBRP,FSQRT, FYL2X, FYL2XP1	Undefined			Roundup or #IS
FCOS, FSIN, FSINCOS, FPTAN	Undefined		0 = source operand within range 1 = source operand out of range	Roundup or #IS (Undefined if C2 = 1)
FABS, FBLD, FCHS, FDECSTP, FILD, FINCSTP, FLD, Load Constants, FSTP (ext. prec.), FXCH, FXTRACT	Undefined			0 or #IS

Table 8-1. Condition Code Interpretation (Contd.)

FLDENV, FRSTOR	Each bit loaded from memory			
FFREE, FLDCW, FCLEX/FNCLEX, FNOP, FSTCW/FNSTCW, FSTENV/FNSTENV, FSTSW/FNSTSW,	Undefined			
FINIT/FNINIT, FSAVE/FNSAVE	0	0	0	0

8.1.3.4 Stack Fault Flag

The stack fault flag (bit 6 of the x87 FPU status word) indicates that stack overflow or stack underflow has occurred with data in the x87 FPU data register stack. The x87 FPU explicitly sets the SF flag when it detects a stack overflow or underflow condition, but it does not explicitly clear the flag when it detects an invalid-arithmetic-operand condition.

When this flag is set, the condition code flag C1 indicates the nature of the fault: overflow (C1 = 1) and underflow (C1 = 0). The SF flag is a "sticky" flag, meaning that after it is set, the processor does not clear it until it is explicitly instructed to do so (for example, by an FINIT/FNINIT, FCLEX/FNCLEX, or FSAVE/FNSAVE instruction).

See Section 8.1.7, "x87 FPU Tag Word," for more information on x87 FPU stack faults.

8.1.4 Branching and Conditional Moves on Condition Codes

The x87 FPU (beginning with the P6 family processors) supports two mechanisms for branching and performing conditional moves according to comparisons of two floating-point values. These mechanism are referred to here as the "old mechanism" and the "new mechanism."

The old mechanism is available in x87 FPU's prior to the P6 family processors and in P6 family processors. This mechanism uses the floating-point compare instructions (FCOM, FCOMP, FCOMPP, FTST, FUCOMPP, FICOM, and FICOMP) to compare two floating-point values and set the condition code flags (C0 through C3) according to the results. The contents of the condition code flags are then copied into the status flags of the EFLAGS register using a two step process (see Figure 8-5):

1. The FSTSW AX instruction moves the x87 FPU status word into the AX register.
2. The SAHF instruction copies the upper 8 bits of the AX register, which includes the condition code flags, into the lower 8 bits of the EFLAGS register.

When the condition code flags have been loaded into the EFLAGS register, conditional jumps or conditional moves can be performed based on the new settings of the status flags in the EFLAGS register.

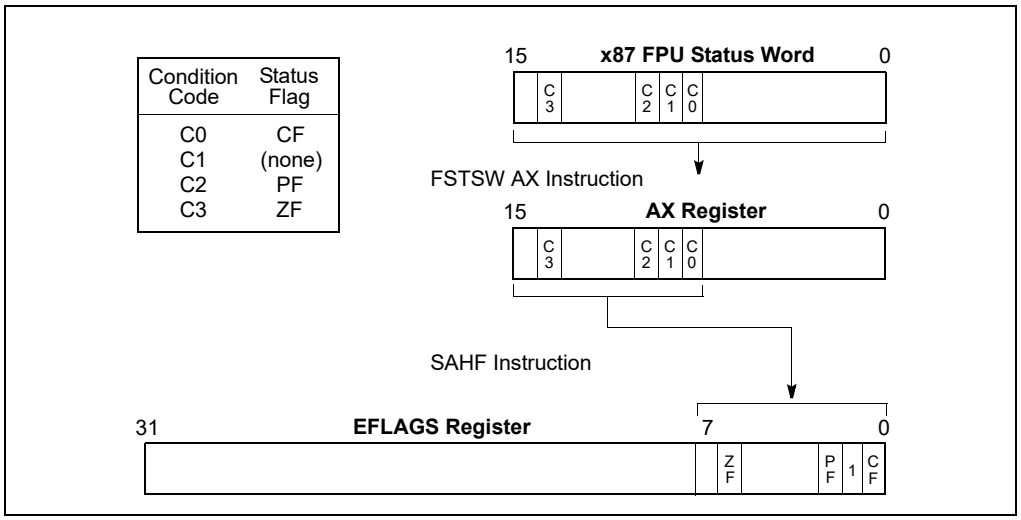


Figure 8-5. Moving the Condition Codes to the EFLAGS Register

The new mechanism is available beginning with the P6 family processors. Using this mechanism, the new floating-point compare and set EFLAGS instructions (FCOMI, FCOMIP, FUCOMI, and FUCOMIP) compare two floating-point values and set the ZF, PF, and CF flags in the EFLAGS register directly. A single instruction thus replaces the three instructions required by the old mechanism.

Note also that the FCMOV*cc* instructions (also new in the P6 family processors) allow conditional moves of floating-point values (values in the x87 FPU data registers) based on the setting of the status flags (ZF, PF, and CF) in the EFLAGS register. These instructions eliminate the need for an IF statement to perform conditional moves of floating-point values.

8.1.5 x87 FPU Control Word

The 16-bit x87 FPU control word (see Figure 8-6) controls the precision of the x87 FPU and rounding method used. It also contains the x87 FPU floating-point exception mask bits. The control word is cached in the x87 FPU control register. The contents of this register can be loaded with the FLDCW instruction and stored in memory with the FSTCW/FNSTCW instructions.

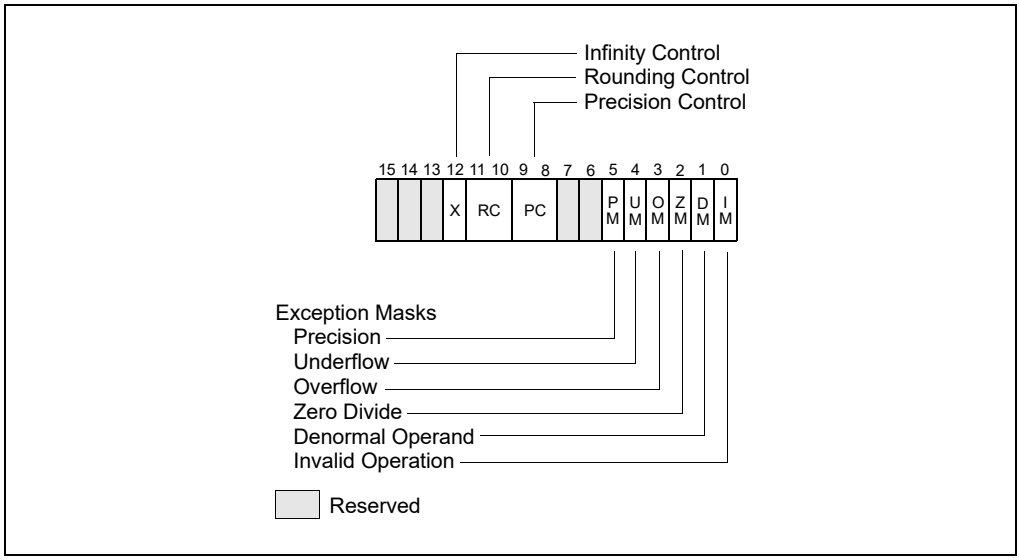


Figure 8-6. x87 FPU Control Word

When the x87 FPU is initialized with either an FINIT/FNINIT or FSAVE/FNSAVE instruction, the x87 FPU control word is set to 037FH, which masks all floating-point exceptions, sets rounding to nearest, and sets the x87 FPU precision to 64 bits.

8.1.5.1 x87 FPU Floating-Point Exception Mask Bits

The exception-flag mask bits (bits 0 through 5 of the x87 FPU control word) mask the 6 floating-point exception flags in the x87 FPU status word. When one of these mask bits is set, its corresponding x87 FPU floating-point exception is blocked from being generated.

8.1.5.2 Precision Control Field

The precision-control (PC) field (bits 8 and 9 of the x87 FPU control word) determines the precision (64, 53, or 24 bits) of floating-point calculations made by the x87 FPU (see Table 8-2). The default precision is double extended precision, which uses the full 64-bit significand available with the double extended-precision floating-point format of the x87 FPU data registers. This setting is best suited for most applications, because it allows applications to take full advantage of the maximum precision available with the x87 FPU data registers.

Table 8-2. Precision Control Field (PC)

Precision	PC Field
Single Precision (24 bits)	00B
Reserved	01B
Double Precision (53 bits)	10B
Double Extended Precision (64 bits)	11B

The double precision and single precision settings reduce the size of the significand to 53 bits and 24 bits, respectively. These settings are provided to support IEEE Standard 754 and to provide compatibility with the specifications of certain existing programming languages. Using these settings nullifies the advantages of the double extended-precision floating-point format's 64-bit significand length. When reduced precision is specified, the rounding of the significand value clears the unused bits on the right to zeros.

The precision-control bits only affect the results of the following floating-point instructions: FADD, FADDP, FIADD, FSUB, FSUBP, FISUB, FSUBR, FSUBRP, FISUBR, FMUL, FMULP, FIMUL, FDIV, FDIVP, FIDIV, FDIVR, FDIVRP, FIDIVR, and FSQRT.

8.1.5.3 Rounding Control Field

The rounding-control (RC) field of the x87 FPU control register (bits 10 and 11) controls how the results of x87 FPU floating-point instructions are rounded. See Section 4.8.4, "Rounding," for a discussion of rounding of floating-point values; See Section 4.8.4.1, "Rounding Control (RC) Fields", for the encodings of the RC field.

8.1.6 Infinity Control Flag

The infinity control flag (bit 12 of the x87 FPU control word) is provided for compatibility with the Intel 287 Math Coprocessor; it is not meaningful for later version x87 FPU coprocessors or IA-32 processors. See Section 4.8.3.3, "Signed Infinities," for information on how the x87 FPUs handle infinity values.

8.1.7 x87 FPU Tag Word

The 16-bit tag word (see Figure 8-7) indicates the contents of each the 8 registers in the x87 FPU data-register stack (one 2-bit tag per register). The tag codes indicate whether a register contains a valid number, zero, or a special floating-point number (NaN, infinity, denormal, or unsupported format), or whether it is empty. The x87 FPU tag word is cached in the x87 FPU in the x87 FPU tag word register. When the x87 FPU is initialized with either an FINIT/FNINIT or FSAVE/FNSAVE instruction, the x87 FPU tag word is set to FFFFH, which marks all the x87 FPU data registers as empty.

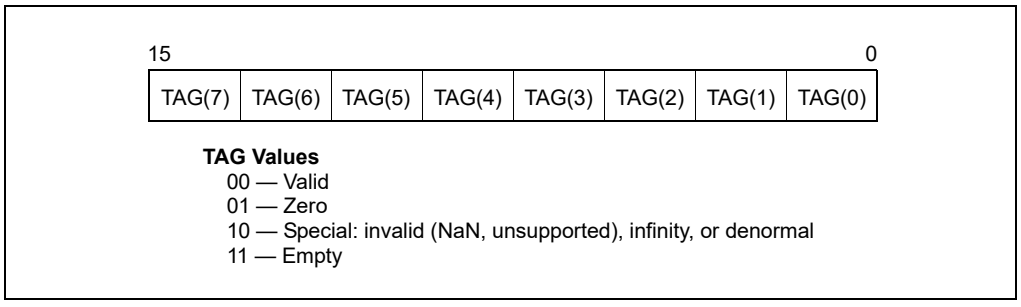


Figure 8-7. x87 FPU Tag Word

Each tag in the x87 FPU tag word corresponds to a physical register (numbers 0 through 7). The current top-of-stack (TOP) pointer stored in the x87 FPU status word can be used to associate tags with registers relative to ST(0).

The x87 FPU uses the tag values to detect stack overflow and underflow conditions (see Section 8.5.1.1, "Stack Overflow or Underflow Exception (#IS)").

Application programs and exception handlers can use this tag information to check the contents of an x87 FPU data register without performing complex decoding of the actual data in the register. To read the tag register, it must be stored in memory using either the FSTENV/FNSTENV or FSAVE/FNSAVE instructions. The location of the tag word in memory after being saved with one of these instructions is shown in Figures 8-9 through 8-12.

Software cannot directly load or modify the tags in the tag register. The FLDENV and FRSTOR instructions load an image of the tag register into the x87 FPU; however, the x87 FPU uses those tag values only to determine if the data registers are empty (11B) or non-empty (00B, 01B, or 10B).

If the tag register image indicates that a data register is empty, the tag in the tag register for that data register is marked empty (11B); if the tag register image indicates that the data register is non-empty, the x87 FPU reads the actual value in the data register and sets the tag for the register accordingly. This action prevents a program from setting the values in the tag register to incorrectly represent the actual contents of non-empty data registers.

8.1.8 x87 FPU Instruction and Data (Operand) Pointers

The x87 FPU stores pointers to the instruction and data (operand) for the last non-control instruction executed. These are the x87 FPU instruction pointer and x87 FPU data (operand) pointers; software can save these pointers to provide state information for exception handlers. The pointers are illustrated in Figure 8-1 (the figure illustrates the pointers as used outside 64-bit mode; see below).

Note that the value in the x87 FPU data pointer is always a pointer to a memory operand. If the last non-control instruction that was executed did not have a memory operand, the value in the data pointer is undefined (reserved). If CPUID.(EAX=07H,ECX=0H):EBX[bit 6] = 1, the data pointer is updated only for x87 non-control instructions that incur unmasked x87 exceptions.

The contents of the x87 FPU instruction and data pointers remain unchanged when any of the following instructions are executed: FCLEX/FNCLEX, FLDCW, FSTCW/FNSTCW, FSTSW/FNSTSW, FSTENV/FNSTENV, FLDENV, and WAIT/FWAIT.

For all the x87 FPUs and Numeric Processor Extensions (NPXs) except the 8087, the x87 FPU instruction pointer points to any prefixes that preceded the instruction. For the 8087, the x87 FPU instruction pointer points only to the actual opcode.

The x87 FPU instruction and data pointers each consists of an offset and a segment selector:

- The x87 FPU Instruction Pointer Offset (FIP) comprises 64 bits on processors that support IA-32e mode; on other processors, it offset comprises 32 bits.
- The x87 FPU Instruction Pointer Selector (FCS) comprises 16 bits.
- The x87 FPU Data Pointer Offset (FDP) comprises 64 bits on processors that support IA-32e mode; on other processors, it offset comprises 32 bits.
- The x87 FPU Data Pointer Selector (FDS) comprises 16 bits.

The pointers are accessed by the FINIT/FNINIT, FLDENV, FRSTOR, FSAVE/FNSAVE, FSTENV/FNSTENV, FXRSTOR, FXSAVE, XRSTOR, XSAVE, and XSAVEOPT instructions as follows:

- FINIT/FNINIT. Each instruction clears FIP, FCS, FDP, and FDS.
- FLDENV, FRSTOR. These instructions use the memory formats given in Figures 8-9 through 8-12:
  - For each of FIP and FDP, each instruction loads the lower 32 bits from memory and clears the upper 32 bits.
  - If CR0.PE = 1, each instruction loads FCS and FDS from memory; otherwise, it clears them.
- FSAVE/FNSAVE, FSTENV/FNSTENV. These instructions use the memory formats given in Figures 8-9 through 8-12.
  - Each instruction saves the lower 32 bits of each FIP and FDP into memory. the upper 32 bits are not saved.
  - If CR0.PE = 1, each instruction saves FCS and FDS into memory. If CPUID.(EAX=07H,ECX=0H):EBX[bit 13] = 1, the processor deprecates FCS and FDS; it saves each as 0000H.

  - After saving these data into memory, FSAVE/FNSAVE clears FIP, FCS, FDP, and FDS.
- FXRSTOR, XRSTOR. These instructions load data from a memory image whose format depend on operating mode and the REX prefix. The memory formats are given in Tables 3-43, 3-46, and 3-47 in Chapter 3, "Instruction Set Reference, A-L," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.
  - Outside of 64-bit mode or if REX.W = 0, the instructions operate as follows:
    - For each of FIP and FDP, each instruction loads the lower 32 bits from memory and clears the upper 32 bits.
    - Each instruction loads FCS and FDS from memory.
  - In 64-bit mode with REX.W = 1, the instructions operate as follows:
    - Each instruction loads FIP and FDP from memory.
    - Each instruction clears FCS and FDS.
- FXSAVE, XSAVE, and XSAVEOPT. These instructions store data into a memory image whose format depend on operating mode and the REX prefix. The memory formats are given in Tables 3-43, 3-46, and 3-47 in Chapter 3, "Instruction Set Reference, A-L," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.
  - Outside of 64-bit mode or if REX.W = 0, the instructions operate as follows:
    - Each instruction saves the lower 32 bits of each of FIP and FDP into memory. The upper 32 bits are not saved.
    - Each instruction saves FCS and FDS into memory. If CPUID.(EAX=07H,ECX=0H):EBX[bit 13] = 1, the processor deprecates FCS and FDS; it saves each as 0000H.
  - In 64-bit mode with REX.W = 1, each instruction saves FIP and FDP into memory. FCS and FDS are not saved.

8.1.9 Last Instruction Opcode

The x87 FPU stores in the 11-bit x87 FPU opcode register (FOP) the opcode of the last x87 non-control instruction executed that incurred an unmasked x87 exception. (This information provides state information for exception handlers.) Only the first and second opcode bytes (after all prefixes) are stored in the x87 FPU opcode register. Figure 8-8 shows the encoding of these two bytes. Since the upper 5 bits of the first opcode byte are the same for all floating-point opcodes (11011B), only the lower 3 bits of this byte are stored in the opcode register.

8.1.9.1 Fopcode Compatibility Sub-mode

Some Pentium 4 and Intel Xeon processors provide program control over the value stored into FOP. Here, bit 2 of the IA32_MISC_ENABLE MSR enables (set) or disables (clear) the fopcode compatibility mode.

If fopcode compatibility mode is enabled, FOP is defined as it had been in previous IA-32 implementations, as the opcode of the last x87 non-control instruction executed (even if that instruction did not incur an unmasked x87 exception).

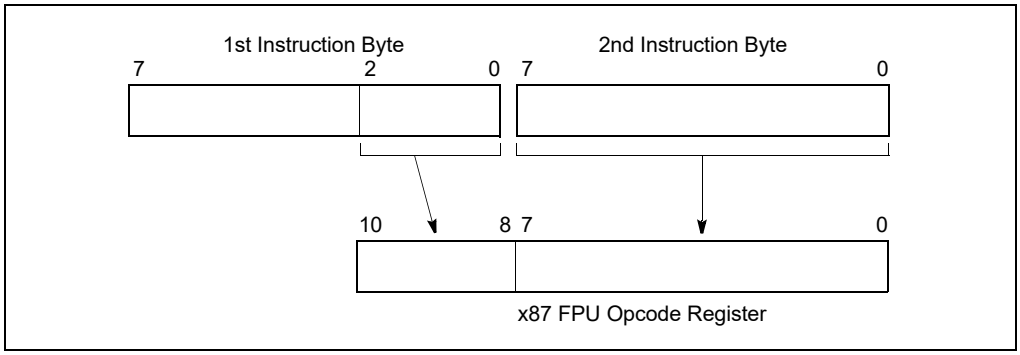


Figure 8-8. Contents of x87 FPU Opcode Registers

The fopcode compatibility mode should be enabled only when x87 FPU floating-point exception handlers are designed to use the fopcode to analyze program performance or restart a program after an exception has been handled.

More recent Intel 64 processors do not support fopcode compatibility mode and do not allow software to set bit 2 of the IA32_MISC_ENABLE MSR.

8.1.10 Saving the x87 FPU's State with FSTENV/FNSTENV and FSAVE/FNSAVE

The FSTENV/FNSTENV and FSAVE/FNSAVE instructions store x87 FPU state information in memory for use by exception handlers and other system and application software. The FSTENV/FNSTENV instruction saves the contents of the status, control, tag, x87 FPU instruction pointer, x87 FPU data pointer, and opcode registers. The FSAVE/FNSAVE instruction stores that information plus the contents of the x87 FPU data registers. Note that the FSAVE/FNSAVE instruction also initializes the x87 FPU to default values (just as the FINIT/FNINIT instruction does) after it has saved the original state of the x87 FPU.

The manner in which this information is stored in memory depends on the operating mode of the processor (protected mode or real-address mode) and on the operand-size attribute in effect (32-bit or 16-bit). See Figures 8-9 through 8-12. In virtual-8086 mode or SMM, the real-address mode formats shown in Figure 8-12 is used. See Chapter 31, "System Management Mode," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C,* for information on using the x87 FPU while in SMM.

The FLDENV and FRSTOR instructions allow x87 FPU state information to be loaded from memory into the x87 FPU. Here, the FLDENV instruction loads only the status, control, tag, x87 FPU instruction pointer, x87 FPU data pointer, and opcode registers, and the FRSTOR instruction loads all the x87 FPU registers, including the x87 FPU stack registers.

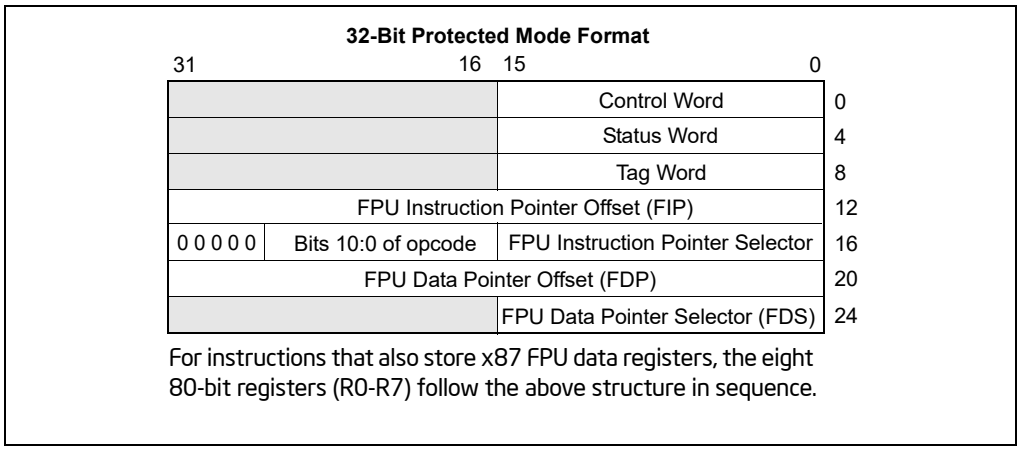


Figure 8-9. Protected Mode x87 FPU State Image in Memory, 32-Bit Format

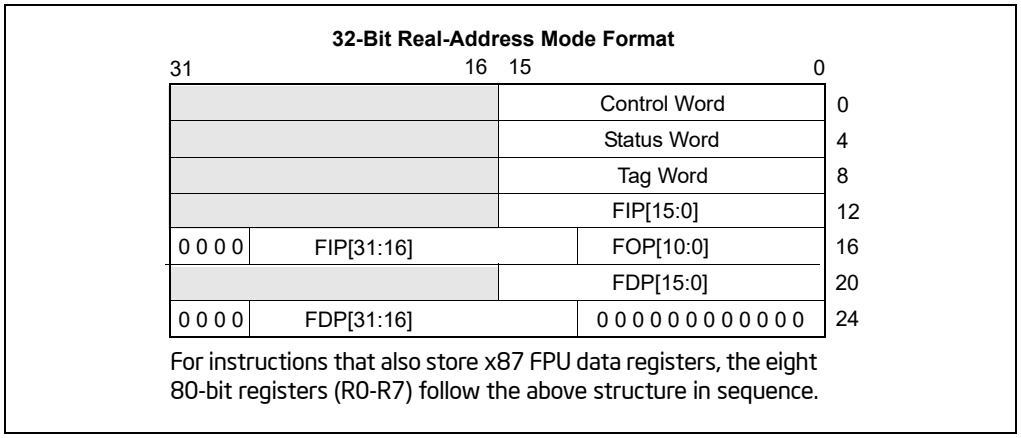


Figure 8-10. Real Mode x87 FPU State Image in Memory, 32-Bit Format

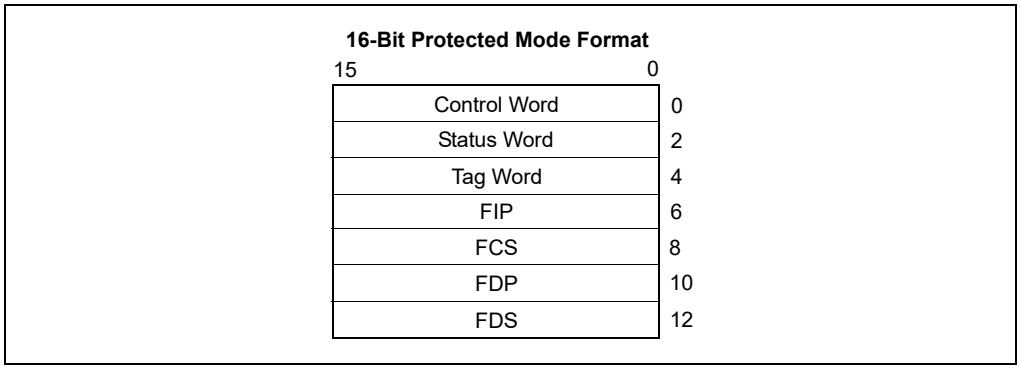


Figure 8-11. Protected Mode x87 FPU State Image in Memory, 16-Bit Format

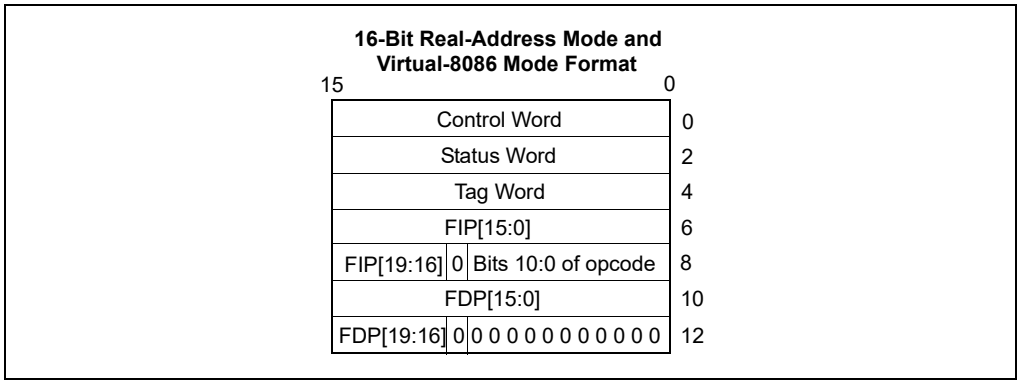


Figure 8-12. Real Mode x87 FPU State Image in Memory, 16-Bit Format

8.1.11 Saving the x87 FPU's State with FXSAVE

The FXSAVE and FXRSTOR instructions save and restore, respectively, the x87 FPU state along with the state of the XMM registers and the MXCSR register. Using the FXSAVE instruction to save the x87 FPU state has two benefits: (1) FXSAVE executes faster than FSAVE, and (2) FXSAVE saves the entire x87 FPU, MMX, and XMM state in one operation. See Section 10.5, "FXSAVE and FXRSTOR Instructions," for additional information about these instructions.

8.2 X87 FPU DATA TYPES

The x87 FPU recognizes and operates on the following seven data types (see Figures 8-13): single-precision floating point, double-precision floating point, double extended-precision floating point, signed word integer, signed doubleword integer, signed quadword integer, and packed BCD decimal integers.

For detailed information about these data types, see Section 4.2.2, "Floating-Point Data Types," Section 4.2.1.2, "Signed Integers," and Section 4.7, "BCD and Packed BCD Integers."

With the exception of the 80-bit double extended-precision floating-point format, all of these data types exist in memory only. When they are loaded into x87 FPU data registers, they are converted into double extended-precision floating-point format and operated on in that format.

Denormal values are also supported in each of the floating-point types, as required by IEEE Standard 754. When a denormal number in single-precision or double-precision floating-point format is used as a source operand and the denormal exception is masked, the x87 FPU automatically **normalizes** the number when it is converted to double extended-precision format.

When stored in memory, the least significant byte of an x87 FPU data-type value is stored at the initial address specified for the value. Successive bytes from the value are then stored in successively higher addresses in memory. The floating-point instructions load and store memory operands using only the initial address of the operand.

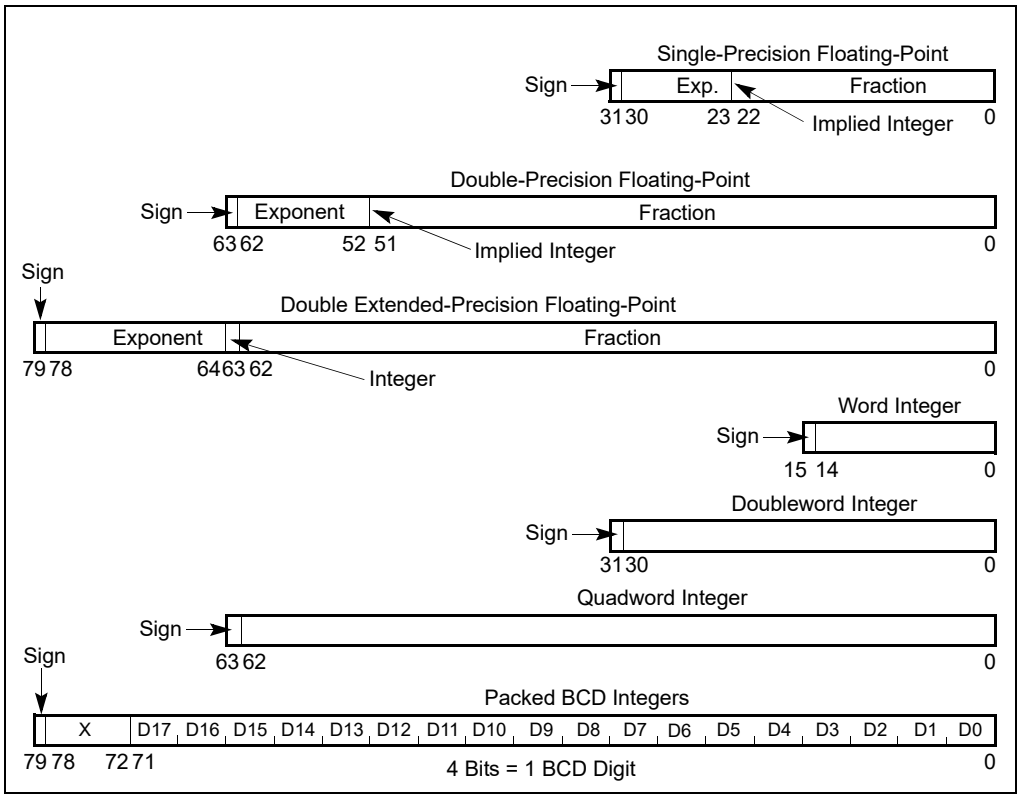


Figure 8-13. x87 FPU Data Type Formats

As a general rule, values should be stored in memory in double-precision format. This format provides sufficient range and precision to return correct results with a minimum of programmer attention. The single-precision format is useful for debugging algorithms, because rounding problems will manifest themselves more quickly in this format. The double extended-precision format is normally reserved for holding intermediate results in the x87 FPU registers and constants. Its extra length is designed to shield final results from the effects of rounding and overflow/underflow in intermediate calculations. However, when an application requires the maximum range and precision of the x87 FPU (for data storage, computations, and results), values can be stored in memory in double extended-precision format.

8.2.1 Indefinites

For each x87 FPU data type, one unique encoding is reserved for representing the special value **indefinite**. The x87 FPU produces indefinite values as responses to some masked floating-point invalid-operation exceptions. See Tables 4-1, 4-3, and 4-4 for the encoding of the integer indefinite, QNaN floating-point indefinite, and packed BCD integer indefinite, respectively.

The binary integer encoding 100..00B represents either of two things, depending on the circumstances of its use:

- The largest negative number supported by the format (-2^{15}, -2^{31}, or -2^{63})
- The **integer indefinite** value

If this encoding is used as a source operand (as in an integer load or integer arithmetic instruction), the x87 FPU interprets it as the largest negative number representable in the format being used. If the x87 FPU detects an invalid operation when storing an integer value in memory with an FIST/FISTP instruction and the invalid-operation exception is masked, the x87 FPU stores the integer indefinite encoding in the destination operand as a masked response to the exception. In situations where the origin of a value with this encoding may be ambiguous, the invalid-operation exception flag can be examined to see if the value was produced as a response to an exception.

8.2.2 Unsupported Double Extended-Precision Floating-Point Encodings and Pseudo-Denormals

The double extended-precision floating-point format permits many encodings that do not fall into any of the categories shown in Table 4-3. Table 8-3 shows these unsupported encodings. Some of these encodings were supported by the Intel 287 math coprocessor; however, most of them are not supported by the Intel 387 math coprocessor and later IA-32 processors. These encodings are no longer supported due to changes made in the final version of IEEE Standard 754 that eliminated these encodings.

Specifically, the categories of encodings formerly known as pseudo-NaNs, pseudo-infinities, and un-normal numbers are not supported and should not be used as operand values. The Intel 387 math coprocessor and later IA-32 processors generate an invalid-operation exception when these encodings are encountered as operands.

Beginning with the Intel 387 math coprocessor, the encodings formerly known as pseudo-denormal numbers are not generated by IA-32 processors. When encountered as operands, however, they are handled correctly, considering the biased exponent as 1 (and the unbiased exponent as -16382); that is, they are treated as denormals and a denormal exception is generated. Pseudo-denormal numbers should not be used as operand values. They are supported by current IA-32 processors (as described here) to support legacy code.

Table 8-3. Unsupported Double Extended-Precision Floating-Point Encodings and Pseudo-Denormals

Class		Sign	Biased Exponent	Significand	
				Integer	Fraction
Positive Pseudo-NaNs	Quiet	0 . 0	11..11 . 11..11	0	11..11 . 10..00
	Signaling	0 . 0	11..11 . 11..11	0	01..11 . 00..01
Positive Floating Point	Pseudo-infinity	0	11..11	0	00..00
	Unnormals	0 . 0	11..10 . 00..01	0	11..11 . 00..00
	Pseudo-denormals	0 . 0	00..00 . 00..00	1	11..11 . 00..00

Table 8-3. Unsupported Double Extended-Precision Floating-Point Encodings and Pseudo-Denormals (Contd.)

Negative Floating Point	Pseudo-denormals	1 . 1	00..00 . 00..00	1	11..11 . 00..00
	Unnormals	1 . 1	11..10 . 00..01	0	11..01 . 00..00
	Pseudo-infinity	1	11..11	0	00..00
Negative Pseudo-NaNs	Signaling	1 . 1	11..11 . 11..11	0	01..11 . 00..01
	Quiet	1 . 1	11..11 . 11..11	0	11..11 . 10..00
			← 15 bits →		← 63 bits →

8.3 X87 FPU INSTRUCTION SET

The floating-point instructions that the x87 FPU supports can be grouped into six functional categories:

- Data transfer instructions
- Basic arithmetic instructions
- Comparison instructions
- Transcendental instructions
- Load constant instructions
- x87 FPU control instructions

See Section , "CPUID.EAX=80000001H:ECX.PREFTEHCHW[bit 8]: if 1 indicates the processor supports the PREFTEHCHW instruction. CPUID.(EAX=07H, ECX=0H):ECX.PREFTEHCHWT1[bit 0]: if 1 indicates the processor supports the PREFTEHCHWT1 instruction.," for a list of the floating-point instructions by category.

The following section briefly describes the instructions in each category. Detailed descriptions of the floating-point instructions are given in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D.*

8.3.1 Escape (ESC) Instructions

All of the instructions in the x87 FPU instruction set fall into a class of instructions known as escape (ESC) instructions. All of these instructions have a common opcode format, where the first byte of the opcode is one of the numbers from D8H through DFH.

8.3.2 x87 FPU Instruction Operands

Most floating-point instructions require one or two operands, located on the x87 FPU data-register stack or in memory. (None of the floating-point instructions accept immediate operands.)

When an operand is located in a data register, it is referenced relative to the ST(0) register (the register at the top of the register stack), rather than by a physical register number. Often the ST(0) register is an implied operand.

Operands in memory can be referenced using the same operand addressing methods described in Section 3.7, "Operand Addressing."

8.3.3 Data Transfer Instructions

The data transfer instructions (see Table 8-4) perform the following operations:

- Load a floating-point, integer, or packed BCD operand from memory into the ST(0) register.
- Store the value in an ST(0) register to memory in floating-point, integer, or packed BCD format.
- Move values between registers in the x87 FPU register stack.

The FLD (load floating point) instruction pushes a floating-point operand from memory onto the top of the x87 FPU data-register stack. If the operand is in single-precision or double-precision floating-point format, it is automatically converted to double extended-precision floating-point format. This instruction can also be used to push the value in a selected x87 FPU data register onto the top of the register stack.

The FILD (load integer) instruction converts an integer operand in memory into double extended-precision floating-point format and pushes the value onto the top of the register stack. The FBLD (load packed decimal) instruction performs the same load operation for a packed BCD operand in memory.

Table 8-4. Data Transfer Instructions

Floating Point		Integer		Packed Decimal	
FLD	Load Floating Point	FILD	Load Integer	FBLD	Load Packed Decimal
FST	Store Floating Point	FIST	Store Integer		
FSTP	Store Floating Point and Pop	FISTP	Store Integer and Pop	FBSTP	Store Packed Decimal and Pop
FXCH	Exchange Register Contents				
FCMOV*cc*	Conditional Move				

The FST (store floating point) and FIST (store integer) instructions store the value in register ST(0) in memory in the destination format (floating point or integer, respectively). Again, the format conversion is carried out automatically.

The FSTP (store floating point and pop), FISTP (store integer and pop), and FBSTP (store packed decimal and pop) instructions store the value in the ST(0) registers into memory in the destination format (floating point, integer, or packed BCD), then performs a **pop** operation on the register stack. A pop operation causes the ST(0) register to be marked empty and the stack pointer (TOP) in the x87 FPU control work to be incremented by 1. The FSTP instruction can also be used to copy the value in the ST(0) register to another x87 FPU register [ST(i)].

The FXCH (exchange register contents) instruction exchanges the value in a selected register in the stack [ST(i)] with the value in ST(0).

The FCMOV*cc* (conditional move) instructions move the value in a selected register in the stack [ST(i)] to register ST(0) if a condition specified with a condition code (*cc*) is satisfied (see Table 8-5). The condition being tested for is represented by the status flags in the EFLAGS register. The condition code mnemonics are appended to the letters "FCMOV" to form the mnemonic for a FCMOV*cc* instruction.

Table 8-5. Floating-Point Conditional Move Instructions

Instruction Mnemonic	Status Flag States	Condition Description
FCMOVB	CF=1	Below
FCMOVNB	CF=0	Not below
FCMOVE	ZF=1	Equal
FCMOVNE	ZF=0	Not equal

Table 8-5. Floating-Point Conditional Move Instructions (Contd.)

Instruction Mnemonic	Status Flag States	Condition Description
FCMOVBE	CF=1 or ZF=1	Below or equal
FCMOVNBE	CF=0 or ZF=0	Not below nor equal
FCMOVU	PF=1	Unordered
FCMOVNU	PF=0	Not unordered

Like the CMOV*cc* instructions, the FCMOV*cc* instructions are useful for optimizing small IF constructions. They also help eliminate branching overhead for IF operations and the possibility of branch mispredictions by the processor.

Software can check if the FCMOV*cc* instructions are supported by checking the processor's feature information with the CPUID instruction.

8.3.4 Load Constant Instructions

The following instructions push commonly used constants onto the top [ST(0)] of the x87 FPU register stack:

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

The constant values have full double extended-precision floating-point precision (64 bits) and are accurate to approximately 19 decimal digits. They are stored internally in a format more precise than double extended-precision floating point. When loading the constant, the x87 FPU rounds the more precise internal constant according to the RC (rounding control) field of the x87 FPU control word. The inexact-result exception (#P) is not generated as a result of this rounding, nor is the C1 flag set in the x87 FPU status word if the value is rounded up. See Section 8.3.8, "Approximation of Pi," for information on the π constant.

8.3.5 Basic Arithmetic Instructions

The following floating-point instructions perform basic arithmetic operations on floating-point numbers. Where applicable, these instructions match IEEE Standard 754:

FADD/FADDP	Add floating point
FIADD	Add integer to floating point
FSUB/FSUBP	Subtract floating point
FISUB	Subtract integer from floating point
FSUBR/FSUBRP	Reverse subtract floating point
FISUBR	Reverse subtract floating point from integer
FMUL/FMULP	Multiply floating point
FIMUL	Multiply integer by floating point
FDIV/FDIVP	Divide floating point
FIDIV	Divide floating point by integer
FDIVR/FDIVRP	Reverse divide
FIDIVR	Reverse divide integer by floating point
FABS	Absolute value
FCHS	Change sign

FSQRT	Square root
FPREM	Partial remainder
FPREM1	IEEE partial remainder
FRNDINT	Round to integral value
FXTRACT	Extract exponent and significand

The add, subtract, multiply and divide instructions operate on the following types of operands:

- Two x87 FPU data registers
- An x87 FPU data register and a floating-point or integer value in memory

See Section 8.1.2, "x87 FPU Data Registers," for a description of how operands are referenced on the data register stack.

Operands in memory can be in single-precision floating-point, double-precision floating-point, word-integer, or doubleword-integer format. They are converted to double extended-precision floating-point format automatically.

Reverse versions of the subtract (FSUBR) and divide (FDIVR) instructions enable efficient coding. For example, the following options are available with the FSUB and FSUBR instructions for operating on values in a specified x87 FPU data register ST(*i*) and the ST(0) register:

FSUB:

ST(0) := ST(0) – ST(*i*)
ST(*i*) := ST(*i*) – ST(0)

FSUBR:

ST(0) := ST(*i*) – ST(0)
ST(*i*) := ST(0) – ST(*i*)

These instructions eliminate the need to exchange values between the ST(0) register and another x87 FPU register to perform a subtraction or division.

The pop versions of the add, subtract, multiply, and divide instructions offer the option of popping the x87 FPU register stack following the arithmetic operation. These instructions operate on values in the ST(*i*) and ST(0) registers, store the result in the ST(*i*) register, and pop the ST(0) register.

The FPREM instruction computes the remainder from the division of two operands in the manner used by the Intel 8087 and Intel 287 math coprocessors; the FPREM1 instruction computes the remainder in the manner specified in IEEE Standard 754.

The FSQRT instruction computes the square root of the source operand.

The FRNDINT instruction returns a floating-point value that is the integral value closest to the source value in the direction of the rounding mode specified in the RC field of the x87 FPU control word.

The FABS, FCHS, and FXTRACT instructions perform convenient arithmetic operations. The FABS instruction produces the absolute value of the source operand. The FCHS instruction changes the sign of the source operand. The FXTRACT instruction separates the source operand into its exponent and fraction and stores each value in a register in floating-point format.

8.3.6 Comparison and Classification Instructions

The following instructions compare or classify floating-point values:

FCOM/FCOMP/FCOMPP Compare floating point and set x87 FPU condition code flags.

FUCOM/FUCOMP/FUCOMPP Unordered compare floating point and set x87 FPU condition code flags.

FICOM/FICOMP Compare integer and set x87 FPU condition code flags.

FCOMI/FCOMIP Compare floating point and set EFLAGS status flags.

FUCOMI/FUCOMIP Unordered compare floating point and set EFLAGS status flags.

FTST Test (compare floating point with 0.0).

FXAM Examine.

Comparison of floating-point values differ from comparison of integers because floating-point values have four (rather than three) mutually exclusive relationships: less than, equal, greater than, and unordered.

The unordered relationship is true when at least one of the two values being compared is a NaN or in an unsupported format. This additional relationship is required because, by definition, NaNs are not numbers, so they cannot have less than, equal, or greater than relationships with other floating-point values.

The FCOM, FCOMP, and FCOMPP instructions compare the value in register ST(0) with a floating-point source operand and set the condition code flags (C0, C2, and C3) in the x87 FPU status word according to the results (see Table 8-6).

If an unordered condition is detected (one or both of the values are NaNs or in an undefined format), a floating-point invalid-operation exception is generated.

The pop versions of the instruction pop the x87 FPU register stack once or twice after the comparison operation is complete.

The FUCOM, FUCOMP, and FUCOMPP instructions operate the same as the FCOM, FCOMP, and FCOMPP instructions. The only difference is that with the FUCOM, FUCOMP, and FUCOMPP instructions, if an unordered condition is detected because one or both of the operands are QNaNs, the floating-point invalid-operation exception is not generated.

Table 8-6. Setting of x87 FPU Condition Code Flags for Floating-Point Number Comparisons

Condition	C3	C2	C0
ST(0) > Source Operand	0	0	0
ST(0) < Source Operand	0	0	1
ST(0) = Source Operand	1	0	0
Unordered	1	1	1

The FICOM and FICOMP instructions also operate the same as the FCOM and FCOMP instructions, except that the source operand is an integer value in memory. The integer value is automatically converted into an double extended-precision floating-point value prior to making the comparison. The FICOMP instruction pops the x87 FPU register stack following the comparison operation.

The FTST instruction performs the same operation as the FCOM instruction, except that the value in register ST(0) is always compared with the value 0.0.

The FCOMI and FCOMIP instructions were introduced into the IA-32 architecture in the P6 family processors. They perform the same comparison as the FCOM and FCOMP instructions, except that they set the status flags (ZF, PF, and CF) in the EFLAGS register to indicate the results of the comparison (see Table 8-7) instead of the x87 FPU condition code flags. The FCOMI and FCOMIP instructions allow condition branch instructions (J*cc*) to be executed directly from the results of their comparison.

Table 8-7. Setting of EFLAGS Status Flags for Floating-Point Number Comparisons

Comparison Results	ZF	PF	CF
ST0 > ST(*i*)	0	0	0
ST0 < ST(*i*)	0	0	1
ST0 = ST(*i*)	1	0	0
Unordered	1	1	1

Software can check if the FCOMI and FCOMIP instructions are supported by checking the processor's feature information with the CPUID instruction.

The FUCOMI and FUCOMIP instructions operate the same as the FCOMI and FCOMIP instructions, except that they do not generate a floating-point invalid-operation exception if the unordered condition is the result of one or both of the operands being a QNaN. The FCOMIP and FUCOMIP instructions pop the x87 FPU register stack following the comparison operation.

The FXAM instruction determines the classification of the floating-point value in the ST(0) register (that is, whether the value is zero, a denormal number, a normal finite number, ∞, a NaN, or an unsupported format) or that the register is empty. It sets the x87 FPU condition code flags to indicate the classification (see "FXAM—Examine" in Chapter 3, "Instruction Set Reference, A-L," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A)*. It also sets the C1 flag to indicate the sign of the value.

8.3.6.1 Branching on the x87 FPU Condition Codes

The processor does not offer any control-flow instructions that branch on the setting of the condition code flags (C0, C2, and C3) in the x87 FPU status word. To branch on the state of these flags, the x87 FPU status word must first be moved to the AX register in the integer unit. The FSTSW AX (store status word) instruction can be used for this purpose. When these flags are in the AX register, the TEST instruction can be used to control conditional branching as follows:

1. Check for an unordered result. Use the TEST instruction to compare the contents of the AX register with the constant 0400H (see Table 8-8). This operation will clear the ZF flag in the EFLAGS register if the condition code flags indicate an unordered result; otherwise, the ZF flag will be set. The JNZ instruction can then be used to transfer control (if necessary) to a procedure for handling unordered operands.

Table 8-8. TEST Instruction Constants for Conditional Branching

Order	Constant	Branch
ST(0) > Source Operand	4500H	JZ
ST(0) < Source Operand	0100H	JNZ
ST(0) = Source Operand	4000H	JNZ
Unordered	0400H	JNZ

2. Check ordered comparison result. Use the constants given in Table 8-8 in the TEST instruction to test for a less than, equal to, or greater than result, then use the corresponding conditional branch instruction to transfer program control to the appropriate procedure or section of code.

If a program or procedure has been thoroughly tested and it incorporates periodic checks for QNaN results, then it is not necessary to check for the unordered result every time a comparison is made.

See Section 8.1.4, "Branching and Conditional Moves on Condition Codes," for another technique for branching on x87 FPU condition codes.

Some non-comparison x87 FPU instructions update the condition code flags in the x87 FPU status word. To ensure that the status word is not altered inadvertently, store it immediately following a comparison operation.

8.3.7 Trigonometric Instructions

The following instructions perform four common trigonometric functions:

FSIN	Sine
FCOS	Cosine
FSINCOS	Sine and cosine
FPTAN	Tangent
FPATAN	Arctangent

These instructions operate on the top one or two registers of the x87 FPU register stack and they return their results to the stack. The source operands for the FSIN, FCOS, FSINCOS, and FPTAN instructions must be given in radians; the source operand for the FPATAN instruction is given in rectangular coordinate units.

The FSINCOS instruction returns both the sine and the cosine of a source operand value. It operates faster than executing the FSIN and FCOS instructions in succession.

The FPATAN instruction computes the arctangent of ST(1) divided by ST(0), returning a result in radians. It is useful for converting rectangular coordinates to polar coordinates.

See Section 8.3.8, "Approximation of Pi" and Section 8.3.10, "Transcendental Instruction Accuracy" for information regarding the accuracy of these instructions.

8.3.8 Approximation of Pi

When the argument (source operand) of a trigonometric function is within the domain of the function, the argument is automatically reduced by the appropriate multiple of 2π through the same reduction mechanism used by the FPREM and FPREM1 instructions. The internal value of π (3.1415926…) that the x87 FPU uses for argument reduction and other computations, denoted as Pi in the expression below. The numerical value of Pi can be written as:

$$Pi = 0.f * 2^2$$

where the fraction f is expressed in binary form as:

f = C90FDAA2 2168C234 C

(The spaces in the fraction above indicate 32-bit boundaries.)

The internal approximation Pi of the value π has a 66 significant bits. Since the exact value of π represented in binary has the next 3 bits equal to 0, it means that Pi is the value of π rounded to nearest-even to 68 bits, and also the value of π rounded toward zero (truncated) to 69 bits.

However, accuracy problems may arise because this relatively short finite approximation Pi of the number π is used for calculating the reduced argument of the trigonometric function approximations in the implementations of FSIN, FCOS, FSINCOS, and FPTAN. Alternately, this means that FSIN (x), FCOS (x), and FPTAN (x) are really approximating the mathematical functions sin (x * π /Pi), cos (x * π / Pi), and tan (x * π / Pi), and not exactly sin (x), cos (x), and tan (x). (Note that FSINCOS is the equivalent of FSIN and FCOS combined together). The period of sin (x * π /Pi) for example is 2* Pi, and not 2π.

See also Section 8.3.10, "Transcendental Instruction Accuracy" for more information on the accuracy of these functions.

8.3.9 Logarithmic, Exponential, and Scale

The following instructions provide two different logarithmic functions, an exponential function and a scale function:

FYL2X	Logarithm
FYL2XP1	Logarithm epsilon
F2XM1	Exponential
FSCALE	Scale

The FYL2X and FYL2XP1 instructions perform two different base 2 logarithmic operations. The FYL2X instruction computes ($y * \log_2 x$). This operation permits the calculation of the log of any base using the following equation:

$$\log_b x = (1/\log_2 b) * \log_2 x$$

The FYL2XP1 instruction computes ($y * \log_2(x + 1)$). This operation provides optimum accuracy for values of x that are close to 0.

The F2XM1 instruction computes ($2^x - 1$). This instruction only operates on source values in the range –1.0 to +1.0.

The FSCALE instruction multiplies the source operand by a power of 2.

8.3.10 Transcendental Instruction Accuracy

New transcendental instruction algorithms were incorporated into the IA-32 architecture beginning with the Pentium processors. These new algorithms (used in transcendental instructions FSIN, FCOS, FSINCOS, FPTAN, FPATAN, F2XM1, FYL2X, and FYL2XP1) allow a higher level of accuracy than was possible in earlier IA-32 processors and x87 math coprocessors. The accuracy of these instructions is measured in terms of **units in the last place** (**ulp**). For a given argument *x*, let *f*(*x*) and *F*(*x*) be the correct and computed (approximate) function values, respectively. The error in ulps is defined to be:

$$error = \left| \frac{f(x) - F(x)}{2^{k-63}} \right|$$

where *k* is an integer such that:

$$1 \le 2^{-k} f(x) < 2.$$

With the Pentium processor and later IA-32 processors, the worst case error on transcendental functions is less than 1 ulp when rounding to the nearest (even) and less than 1.5 ulps when rounding in other modes. The functions are guaranteed to be monotonic, with respect to the input operands, throughout the domain supported by the instruction.

However, for FSIN, FCOS, FSINCOS, and FPTAN which approximate periodic trigonometric functions, the previous statement about maximum ulp errors is true only when these instructions are applied to reduced argument (see Section 8.3.8, "Approximation of Pi"). This is due to the fact that only 66 significant bits are retained in the finite approximation Pi of the number π (3.1415926…), used internally for calculating the reduced argument in FSIN, FCOS, FSINCOS, and FPTAN. This approximation of π is not always sufficiently accurate for good argument reduction.

For single precision, the argument of FSIN, FCOS, FSINCOS, and FPTAN must exceed 200,000 radians in order for the error of the result to exceed 1 ulp when rounding to the nearest (even), or 1.5 ulps when rounding in other (directed) rounding modes.

For double and double-extended precision, the ulp errors will grow above these thresholds for arguments much smaller in magnitude. The ulp errors increase significantly when the argument approaches the value of π (or Pi) for FSIN, and when it approaches π/2(or Pi/2) for FCOS, FSINCOS, and FPTAN.

For all three IEEE precisions supported (32-bit single precision, 64-bit double precision, and 80-bit double-extended precision), applying FSIN, FCOS, FSINCOS, or FPTAN to arguments larger than a certain value can lead to reduced arguments (calculated internally) that are inaccurate or even very inaccurate in some cases. This leads to equally inaccurate approximations of the corresponding mathematical functions. In particular, arguments that are close to certain values will lose significance when reduced, leading to increased relative (and ulp) errors in the results of FSIN, FCOS, FSINCOS, and FPTAN. These values are:

- any non-zero multiple of π for FSIN,
- any multiple of π, plus π/2 for FCOS, and
- any non-zero multiple of π/2 for FSINCOS and FPTAN.

If the arguments passed to FSIN, FCOS, FSINCOS, and FPTAN are not close to these values then even the finite approximation Pi of π used internally for argument reduction will allow for results that have good accuracy.

Therefore, in order to avoid such errors it is recommended to perform accurate argument reduction in software, and to apply FSIN, FCOS, FSINCOS, and FPTAN to reduced arguments only. Regardless of the target precision (single, double, or double-extended), it is safe to reduce the argument to a value smaller in absolute value than about 3π/4 for FSIN, and smaller than about 3π/8 for FCOS, FSINCOS, and FPTAN.

The thresholds shown above are not exact. For example, accuracy measurements show that the double-extended precision result of FSIN will not have errors larger than 0.72 ulp for |x| < 2.82 (so |x| < 3π/4 will ensure good accuracy, as 3π/4 < 2.82). On the same interval, double precision results from FSIN will have errors at most slightly larger than 0.5 ulp, and single precision results will be correctly rounded in the vast majority of cases.

Likewise, the double-extended precision result of FCOS will not have errors larger than 0.82 ulp for |x| < 1.31 (so |x| < 3π/8 will ensure good accuracy, as 3π/8 < 1.31). On the same interval, double precision results from FCOS

will have errors at most slightly larger than 0.5 ulp, and single precision results will be correctly rounded in the vast majority of cases.

FSINCOS behaves similarly to FSIN and FCOS, combined as a pair.

Finally, the double-extended precision result of FPTAN will not have errors larger than 0.78 ulp for $|x| < 1.25$ (so $|x| < 3\pi/8$ will ensure good accuracy, as $3\pi/8 < 1.25$). On the same interval, double precision results from FPTAN will have errors at most slightly larger than 0.5 ulp, and single precision results will be correctly rounded in the vast majority of cases.

A recommended alternative in order to avoid the accuracy issues that might be caused by FSIN, FCOS, FSINCOS, and FPTAN, is to use good quality mathematical library implementations of the sin, cos, sincos, and tan functions, for example those from the Intel® Math Library available in the Intel® Compiler.

The instructions FYL2X and FYL2XP1 are two operand instructions and are guaranteed to be within 1 ulp only when y equals 1. When y is not equal to 1, the maximum ulp error is always within 1.35 ulps in round to nearest mode. (For the two operand functions, monotonicity was proved by holding one of the operands constant.)

8.3.11 x87 FPU Control Instructions

The following instructions control the state and modes of operation of the x87 FPU. They also allow the status of the x87 FPU to be examined:

FINIT/FNINIT	Initialize x87 FPU
FLDCW	Load x87 FPU control word
FSTCW/FNSTCW	Store x87 FPU control word
FSTSW/FNSTSW	Store x87 FPU status word
FCLEX/FNCLEX	Clear x87 FPU exception flags
FLDENV	Load x87 FPU environment
FSTENV/FNSTENV	Store x87 FPU environment
FRSTOR	Restore x87 FPU state
FSAVE/FNSAVE	Save x87 FPU state
FINCSTP	Increment x87 FPU register stack pointer
FDECSTP	Decrement x87 FPU register stack pointer
FFREE	Free x87 FPU register
FNOP	No operation
WAIT/FWAIT	Check for and handle pending unmasked x87 FPU exceptions

The FINIT/FNINIT instructions initialize the x87 FPU and its internal registers to default values.

The FLDCW instructions loads the x87 FPU control word register with a value from memory. The FSTCW/FNSTCW and FSTSW/FNSTSW instructions store the x87 FPU control and status words, respectively, in memory (or for an FSTSW/FNSTSW instruction in a general-purpose register).

The FSTENV/FNSTENV and FSAVE/FNSAVE instructions save the x87 FPU environment and state, respectively, in memory. The x87 FPU environment includes all the x87 FPU's control and status registers; the x87 FPU state includes the x87 FPU environment and the data registers in the x87 FPU register stack. (The FSAVE/FNSAVE instruction also initializes the x87 FPU to default values, like the FINIT/FNINIT instruction, after it saves the original state of the x87 FPU.)

The FLDENV and FRSTOR instructions load the x87 FPU environment and state, respectively, from memory into the x87 FPU. These instructions are commonly used when switching tasks or contexts.

The WAIT/FWAIT instructions are synchronization instructions. (They are actually mnemonics for the same opcode.) These instructions check the x87 FPU status word for pending unmasked x87 FPU exceptions. If any pending unmasked x87 FPU exceptions are found, they are handled before the processor resumes execution of the instructions (integer, floating-point, or system instruction) in the instruction stream. The WAIT/FWAIT instructions

are provided to allow synchronization of instruction execution between the x87 FPU and the processor's integer unit. See Section 8.6, "x87 FPU Exception Synchronization," for more information on the use of the WAIT/FWAIT instructions.

8.3.12 Waiting vs. Non-waiting Instructions

All of the x87 FPU instructions except a few special control instructions perform a wait operation (similar to the WAIT/FWAIT instructions), to check for and handle pending unmasked x87 FPU floating-point exceptions, before they perform their primary operation (such as adding two floating-point numbers). These instructions are called **waiting** instructions. Some of the x87 FPU control instructions, such as FSTSW/FNSTSW, have both a waiting and a non-waiting version. The waiting version (with the "F" prefix) executes a wait operation before it performs its primary operation; whereas, the non-waiting version (with the "FN" prefix) ignores pending unmasked exceptions.

Non-waiting instructions allow software to save the current x87 FPU state without first handling pending exceptions or to reset or reinitialize the x87 FPU without regard for pending exceptions.

NOTES

When operating a Pentium or Intel486 processor in MS-DOS compatibility mode, it is possible (under unusual circumstances) for a non-waiting instruction to be interrupted prior to being executed to handle a pending x87 FPU exception.

When operating a P6 family, Pentium 4, or Intel Xeon processor in MS-DOS compatibility mode, non-waiting instructions can not be interrupted in this way.

8.3.13 Unsupported x87 FPU Instructions

The Intel 8087 instructions FENI and FDISI and the Intel 287 math coprocessor instruction FSETPM perform no function in the Intel 387 math coprocessor and later IA-32 processors. If these opcodes are detected in the instruction stream, the x87 FPU performs no specific operation and no internal x87 FPU states are affected.

8.4 X87 FPU FLOATING-POINT EXCEPTION HANDLING

The x87 FPU detects the six classes of exception conditions described in Section 4.9, "Overview of Floating-Point Exceptions":

- Invalid operation (#I), with two subclasses:
  - Stack overflow or underflow (#IS)
  - Invalid arithmetic operation (#IA)
- Denormalized operand (#D)
- Divide-by-zero (#Z)
- Numeric overflow (#O)
- Numeric underflow (#U)
- Inexact result (precision) (#P)

Each of the six exception classes has a corresponding flag bit in the x87 FPU status word and a mask bit in the x87 FPU control word (see Section 8.1.3, "x87 FPU Status Register," and Section 8.1.5, "x87 FPU Control Word," respectively). In addition, the exception summary (ES) flag in the status word indicates when one or more unmasked exceptions has been detected. The stack fault (SF) flag (also in the status word) distinguishes between the two types of invalid-operation exceptions.

The mask bits can be set with FLDCW, FRSTOR, or FXRSTOR; they can be read with either FSTCW/FNSTCW, FSAVE/FNSAVE, or FXSAVE. The flag bits can be read with the FSTSW/FNSTSW, FSAVE/FNSAVE, or FXSAVE instruction.

NOTE

Section 4.9.1, "Floating-Point Exception Conditions," provides a general overview of how the IA-32 processor detects and handles the various classes of floating-point exceptions. This information pertains to x87 FPU as well as SSE/SSE2/SSE3 extensions.

The following sections give specific information about how the x87 FPU handles floating-point exceptions that are unique to the x87 FPU.

8.4.1 Arithmetic vs. Non-arithmetic Instructions

When dealing with floating-point exceptions, it is useful to distinguish between **arithmetic instructions** and **non-arithmetic instructions**. Non-arithmetic instructions have no operands or do not make substantial changes to their operands. Arithmetic instructions do make significant changes to their operands; in particular, they make changes that could result in floating-point exceptions being signaled. Table 8-9 lists the non-arithmetic and arithmetic instructions. It should be noted that some non-arithmetic instructions can signal a floating-point stack (fault) exception, but this exception is not the result of an operation on an operand.

Table 8-9. Arithmetic and Non-arithmetic Instructions

Non-arithmetic Instructions	Arithmetic Instructions
FABS	F2XM1
FCHS	FADD/FADDP
FCLEX	FBLD
FDECSTP	FBSTP
FFREE	FCOM/FCOMP/FCOMPP
FINCSTP	FCOS
FINIT/FNINIT	FDIV/FDIVP/FDIVR/FDIVRP
FLD (register-to-register)	FIADD
FLD (extended format from memory)	FICOM/FICOMP
FLD constant	FIDIV/FIDIVR
FLDCW	FILD
FLDENV	FIMUL
FNOP	FIST/FISTP[1]
FRSTOR	FISUB/FISUBR
FSAVE/FNSAVE	FLD (single and double)
FST/FSTP (register-to-register)	FMUL/FMULP
FSTP (extended format to memory)	FPATAN
FSTCW/FNSTCW	FPREM/FPREM1
FSTENV/FNSTENV	FPTAN
FSTSW/FNSTSW	FRNDINT
WAIT/FWAIT	FSCALE
FXAM	FSIN
FXCH	FSINCOS
	FSQRT
	FST/FSTP (single and double)
	FSUB/FSUBP/FSUBR/FSUBRP

Table 8-9. Arithmetic and Non-arithmetic Instructions (Contd.)

Non-arithmetic Instructions	Arithmetic Instructions
	FTST
	FUCOM/FUCOMP/FUCOMPP
	FXTRACT
	FYL2X/FYL2XP1

NOTE:

1. The FISTTP instruction in SSE3 is an arithmetic x87 FPU instruction.

8.5 X87 FPU FLOATING-POINT EXCEPTION CONDITIONS

The following sections describe the various conditions that cause a floating-point exception to be generated by the x87 FPU and the masked response of the x87 FPU when these conditions are detected. *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A & 2B,* list the floating-point exceptions that can be signaled for each floating-point instruction.

See Section 4.9.2, "Floating-Point Exception Priority," for a description of the rules for exception precedence when more than one floating-point exception condition is detected for an instruction.

8.5.1 Invalid Operation Exception

The floating-point invalid-operation exception occurs in response to two sub-classes of operations:

- Stack overflow or underflow (#IS)
- Invalid arithmetic operand (#IA)

The flag for this exception (IE) is bit 0 of the x87 FPU status word, and the mask bit (IM) is bit 0 of the x87 FPU control word. The stack fault flag (SF) of the x87 FPU status word indicates the type of operation that caused the exception. When the SF flag is set to 1, a stack operation has resulted in stack overflow or underflow; when the flag is cleared to 0, an arithmetic instruction has encountered an invalid operand. Note that the x87 FPU explicitly sets the SF flag when it detects a stack overflow or underflow condition, but it does not explicitly clear the flag when it detects an invalid-arithmetic-operand condition. As a result, the state of the SF flag can be 1 following an invalid-arithmetic-operation exception, if it was not cleared from the last time a stack overflow or underflow condition occurred. See Section 8.1.3.4, "Stack Fault Flag," for more information about the SF flag.

8.5.1.1 Stack Overflow or Underflow Exception (#IS)

The x87 FPU tag word keeps track of the contents of the registers in the x87 FPU register stack (see Section 8.1.7, "x87 FPU Tag Word"). It then uses this information to detect two different types of stack faults:

- **Stack overflow** — An instruction attempts to load a non-empty x87 FPU register from memory. A non-empty register is defined as a register containing a zero (tag value of 01), a valid value (tag value of 00), or a special value (tag value of 10).
- **Stack underflow** — An instruction references an empty x87 FPU register as a source operand, including attempting to write the contents of an empty register to memory. An empty register has a tag value of 11.

NOTES

The term stack overflow originates from the situation where the program has loaded (pushed) eight values from memory onto the x87 FPU register stack and the next value pushed on the stack causes a stack wraparound to a register that already contains a value.

The term stack underflow originates from the opposite situation. Here, a program has stored (popped) eight values from the x87 FPU register stack to memory and the next value popped from the stack causes stack wraparound to an empty register.

When the x87 FPU detects stack overflow or underflow, it sets the IE flag (bit 0) and the SF flag (bit 6) in the x87 FPU status word to 1. It then sets condition-code flag C1 (bit 9) in the x87 FPU status word to 1 if stack overflow occurred or to 0 if stack underflow occurred.

If the invalid-operation exception is masked, the x87 FPU returns the floating point, integer, or packed decimal integer indefinite value to the destination operand, depending on the instruction being executed. This value overwrites the destination register or memory location specified by the instruction.

If the invalid-operation exception is not masked, a software exception handler is invoked (see Section 8.7, "Handling x87 FPU Exceptions in Software") and the top-of-stack pointer (TOP) and source operands remain unchanged.

8.5.1.2 Invalid Arithmetic Operand Exception (#IA)

The x87 FPU is able to detect a variety of invalid arithmetic operations that can be coded in a program. These operations are listed in Table 8-10. (This list includes the invalid operations defined in IEEE Standard 754.)

When the x87 FPU detects an invalid arithmetic operand, it sets the IE flag (bit 0) in the x87 FPU status word to 1. If the invalid-operation exception is masked, the x87 FPU then returns an indefinite value or QNaN to the destination operand and/or sets the floating-point condition codes as shown in Table 8-10. If the invalid-operation exception is not masked, a software exception handler is invoked (see Section 8.7, "Handling x87 FPU Exceptions in Software") and the top-of-stack pointer (TOP) and source operands remain unchanged.

Table 8-10. Invalid Arithmetic Operations and the Masked Responses to Them

Condition	Masked Response
Any arithmetic operation on an operand that is in an unsupported format.	Return the QNaN floating-point indefinite value to the destination operand.
Any arithmetic operation on a SNaN.	Return a QNaN to the destination operand (see Table 4-7).
Ordered compare and test operations: one or both operands are NaNs.	Set the condition code flags (C0, C2, and C3) in the x87 FPU status word or the CF, PF, and ZF flags in the EFLAGS register to 111B (not comparable).
Addition: operands are opposite-signed infinities. Subtraction: operands are like-signed infinities.	Return the QNaN floating-point indefinite value to the destination operand.
Multiplication: ∞ by 0; 0 by ∞.	Return the QNaN floating-point indefinite value to the destination operand.
Division: ∞ by ∞; 0 by 0.	Return the QNaN floating-point indefinite value to the destination operand.
Remainder instructions FPREM, FPREM1: modulus (divisor) is 0 or dividend is ∞.	Return the QNaN floating-point indefinite; clear condition code flag C2 to 0.
Trigonometric instructions FCOS, FPTAN, FSIN, FSINCOS: source operand is ∞.	Return the QNaN floating-point indefinite; clear condition code flag C2 to 0.
FSQRT: negative operand (except FSQRT (–0) = –0); FYL2X: negative operand (except FYL2X (–0) = –∞); FYL2XP1: operand more negative than –1.	Return the QNaN floating-point indefinite value to the destination operand.
FBSTP: Converted value cannot be represented in 18 decimal digits, or source value is an SNaN, QNaN, $\pm\infty$, or in an unsupported format.	Store packed BCD integer indefinite value in the destination operand.
FIST/FISTP: Converted value exceeds representable integer range of the destination operand, or source value is an SNaN, QNaN, $\pm\infty$, or in an unsupported format.	Store integer indefinite value in the destination operand.
FXCH: one or both registers are tagged empty.	Load empty registers with the QNaN floating-point indefinite value, then perform the exchange.

Normally, when one or both of the source operands is a QNaN (and neither is an SNaN or in an unsupported format), an invalid-operand exception is not generated. An exception to this rule is most of the compare instructions (such as the FCOM and FCOMI instructions) and the floating-point to integer conversion instructions (FIST/FISTP and FBSTP). With these instructions, a QNaN source operand will generate an invalid-operand exception.

8.5.2 Denormal Operand Exception (#D)

The x87 FPU signals the denormal-operand exception under the following conditions:

- If an arithmetic instruction attempts to operate on a denormal operand (see Section 4.8.3.2, "Normalized and Denormalized Finite Numbers").
- If an attempt is made to load a denormal single-precision or double-precision floating-point value into an x87 FPU register. (If the denormal value being loaded is a double extended-precision floating-point value, the denormal-operand exception is not reported.)

The flag (DE) for this exception is bit 1 of the x87 FPU status word, and the mask bit (DM) is bit 1 of the x87 FPU control word.

When a denormal-operand exception occurs and the exception is masked, the x87 FPU sets the DE flag, then proceeds with the instruction. The denormal operand in single- or double-precision floating-point format is automatically normalized when converted to the double extended-precision floating-point format. Subsequent operations will benefit from the additional precision of the internal double extended-precision floating-point format.

When a denormal-operand exception occurs and the exception is not masked, the DE flag is set and a software exception handler is invoked (see Section 8.7, "Handling x87 FPU Exceptions in Software"). The top-of-stack pointer (TOP) and source operands remain unchanged.

For additional information about the denormal-operation exception, see Section 4.9.1.2, "Denormal Operand Exception (#D)."

8.5.3 Divide-By-Zero Exception (#Z)

The x87 FPU reports a floating-point divide-by-zero exception whenever an instruction attempts to divide a finite non-zero operand by 0. The flag (ZE) for this exception is bit 2 of the x87 FPU status word, and the mask bit (ZM) is bit 2 of the x87 FPU control word. The FDIV, FDIVP, FDIVR, FDIVRP, FIDIV, and FIDIVR instructions and the other instructions that perform division internally (FYL2X and FXTRACT) can report the divide-by-zero exception.

When a divide-by-zero exception occurs and the exception is masked, the x87 FPU sets the ZE flag and returns the values shown in Table 8-10. If the divide-by-zero exception is not masked, the ZE flag is set, a software exception handler is invoked (see Section 8.7, "Handling x87 FPU Exceptions in Software"), and the top-of-stack pointer (TOP) and source operands remain unchanged.

Table 8-11. Divide-By-Zero Conditions and the Masked Responses to Them

Condition	Masked Response
Divide or reverse divide operation with a 0 divisor.	Returns an ∞ signed with the exclusive OR of the sign of the two operands to the destination operand.
FYL2X instruction.	Returns an ∞ signed with the opposite sign of the non-zero operand to the destination operand.
FXTRACT instruction.	ST(1) is set to –∞; ST(0) is set to 0 with the same sign as the source operand.

8.5.4 Numeric Overflow Exception (#O)

The x87 FPU reports a floating-point numeric overflow exception (#O) whenever the rounded result of an arithmetic instruction exceeds the largest allowable finite value that will fit into the floating-point format of the destination operand. (See Section 4.9.1.4, "Numeric Overflow Exception (#O)," for additional information about the numeric overflow exception.)

When using the x87 FPU, numeric overflow can occur on arithmetic operations where the result is stored in an x87 FPU data register. It can also occur on store floating-point operations (using the FST and FSTP instructions), where a within-range value in a data register is stored in memory in a single-precision or double-precision floating-point format. The numeric overflow exception cannot occur when storing values in an integer or BCD integer format. Instead, the invalid-arithmetic-operand exception is signaled.

The flag (OE) for the numeric-overflow exception is bit 3 of the x87 FPU status word, and the mask bit (OM) is bit 3 of the x87 FPU control word.

When a numeric-overflow exception occurs and the exception is masked, the x87 FPU sets the OE flag and returns one of the values shown in Table 4-10. The value returned depends on the current rounding mode of the x87 FPU (see Section 8.1.5.3, "Rounding Control Field").

The action that the x87 FPU takes when numeric overflow occurs and the numeric-overflow exception is not masked, depends on whether the instruction is supposed to store the result in memory or on the register stack.

- **Destination is a memory location —** The OE flag is set and a software exception handler is invoked (see Section 8.7, "Handling x87 FPU Exceptions in Software"). The top-of-stack pointer (TOP) and source and destination operands remain unchanged. Because the data in the stack is in double extended-precision format, the exception handler has the option either of re-executing the store instruction after proper adjustment of the operand or of rounding the significand on the stack to the destination's precision as the standard requires. The exception handler should ultimately store a value into the destination location in memory if the program is to continue.
- **Destination is the register stack —** The significand of the result is rounded according to current settings of the precision and rounding control bits in the x87 FPU control word and the exponent of the result is adjusted by dividing it by 2^{24576}. (For instructions not affected by the precision field, the significand is rounded to double-extended precision.) The resulting value is stored in the destination operand. Condition code bit C1 in the x87 FPU status word (called in this situation the "round-up bit") is set if the significand was rounded upward and cleared if the result was rounded toward 0. After the result is stored, the OE flag is set and a software exception handler is invoked. The scaling bias value 24,576 is equal to $3 * 2^{13}$. Biasing the exponent by 24,576 normally translates the number as nearly as possible to the middle of the double extended-precision floating-point exponent range so that, if desired, it can be used in subsequent scaled operations with less risk of causing further exceptions.

  When using the FSCALE instruction, massive overflow can occur, where the result is too large to be represented, even with a bias-adjusted exponent. Here, if overflow occurs again, after the result has been biased, a properly signed ∞ is stored in the destination operand.

8.5.5 Numeric Underflow Exception (#U)

The x87 FPU detects a potential floating-point numeric underflow condition whenever the result of an arithmetic instruction is non-zero and tiny; that is, the magnitude of the rounded result with unbounded exponent is non-zero and less than the smallest possible normalized, finite value that will fit into the floating-point format of the destination operand. (See Section 4.9.1.5, "Numeric Underflow Exception (#U)," for additional information about the numeric underflow exception.)

Like numeric overflow, numeric underflow can occur on arithmetic operations where the result is stored in an x87 FPU data register. It can also occur on store floating-point operations (with the FST and FSTP instructions), where a within-range value in a data register is stored in memory in the smaller single-precision or double-precision floating-point formats. A numeric underflow exception cannot occur when storing values in an integer or BCD integer format, because a value with magnitude less than 1 is always rounded to an integral value of 0 or 1, depending on the rounding mode in effect.

The flag (UE) for the numeric-underflow exception is bit 4 of the x87 FPU status word, and the mask bit (UM) is bit 4 of the x87 FPU control word.

When a numeric-underflow condition occurs and the exception is masked, the x87 FPU performs the operation described in Section 4.9.1.5, "Numeric Underflow Exception (#U)."

When the exception is not masked, the action of the x87 FPU depends on whether the instruction is supposed to store the result in a memory location or on the x87 FPU resister stack.

- **Destination is a memory location —** (Can occur only with a store instruction.) The UE flag is set and a software exception handler is invoked (see Section 8.7, "Handling x87 FPU Exceptions in Software"). The top-of-stack pointer (TOP) and source and destination operands remain unchanged, and no result is stored in memory.
  Because the data in the stack is in double extended-precision format, the exception handler has the option either of re-exchanges the store instruction after proper adjustment of the operand or of rounding the significand on the stack to the destination's precision as the standard requires. The exception handler should ultimately store a value into the destination location in memory if the program is to continue.
- **Destination is the register stack —** The significand of the result is rounded according to current settings of the precision and rounding control bits in the x87 FPU control word and the exponent of the result is adjusted by multiplying it by 2^{24576}. (For instructions not affected by the precision field, the significand is rounded to double extended precision.) The resulting value is stored in the destination operand. Condition code bit C1 in the x87 FPU status register (acting here as a "round-up bit") is set if the significand was rounded upward and cleared if the result was rounded toward 0. After the result is stored, the UE flag is set and a software exception handler is invoked. The scaling bias value 24,576 is the same as is used for the overflow exception and has the same effect, which is to translate the result as nearly as possible to the middle of the double extended-precision floating-point exponent range.

  When using the FSCALE instruction, massive underflow can occur, where the magnitude of the result is too small to be represented, even with a bias-adjusted exponent. Here, if underflow occurs again after the result has been biased, a properly signed 0 is stored in the destination operand.

8.5.6 Inexact-Result (Precision) Exception (#P)

The inexact-result exception (also called the precision exception) occurs if the result of an operation is not exactly representable in the destination format. (See Section 4.9.1.6, "Inexact-Result (Precision) Exception (#P)," for additional information about the numeric overflow exception.) Note that the transcendental instructions (FSIN, FCOS, FSINCOS, FPTAN, FPATAN, F2XM1, FYL2X, and FYL2XP1) by nature produce inexact results.

The inexact-result exception flag (PE) is bit 5 of the x87 FPU status word, and the mask bit (PM) is bit 5 of the x87 FPU control word.

If the inexact-result exception is masked when an inexact-result condition occurs and a numeric overflow or underflow condition has not occurred, the x87 FPU handles the exception as describe in Section 4.9.1.6, "Inexact-Result (Precision) Exception (#P)," with one additional action. The C1 (round-up) bit in the x87 FPU status word is set to indicate whether the inexact result was rounded up (C1 is set) or "not rounded up" (C1 is cleared). In the "not rounded up" case, the least-significant bits of the inexact result are truncated so that the result fits in the destination format.

If the inexact-result exception is not masked when an inexact result occurs and numeric overflow or underflow has not occurred, the x87 FPU handles the exception as described in the previous paragraph and, in addition, invokes a software exception handler.

If an inexact result occurs in conjunction with numeric overflow or underflow, the x87 FPU carries out one of the following operations:

- If an inexact result occurs in conjunction with masked overflow or underflow, the OE or UE flag and the PE flag are set and the result is stored as described for the overflow or underflow exceptions (see Section 8.5.4, "Numeric Overflow Exception (#O)," or Section 8.5.5, "Numeric Underflow Exception (#U)"). If the inexact result exception is unmasked, the x87 FPU also invokes a software exception handler.
- If an inexact result occurs in conjunction with unmasked overflow or underflow and the destination operand is a register, the OE or UE flag and the PE flag are set, the result is stored as described for the overflow or underflow exceptions (see Section 8.5.4, "Numeric Overflow Exception (#O)," or Section 8.5.5, "Numeric Underflow Exception (#U)") and a software exception handler is invoked.

If an unmasked numeric overflow or underflow exception occurs and the destination operand is a memory location (which can happen only for a floating-point store), the inexact-result condition is not reported and the C1 flag is cleared.

8.6 X87 FPU EXCEPTION SYNCHRONIZATION

Because the integer unit and x87 FPU are separate execution units, it is possible for the processor to execute floating-point, integer, and system instructions concurrently. No special programming techniques are required to gain the advantages of concurrent execution. (Floating-point instructions are placed in the instruction stream along with the integer and system instructions.) However, concurrent execution can cause problems for floating-point exception handlers.

This problem is related to the way the x87 FPU signals the existence of unmasked floating-point exceptions. (Special exception synchronization is not required for masked floating-point exceptions, because the x87 FPU always returns a masked result to the destination operand.)

When a floating-point exception is unmasked and the exception condition occurs, the x87 FPU stops further execution of the floating-point instruction and signals the exception event. On the next occurrence of a floating-point instruction or a WAIT/FWAIT instruction in the instruction stream, the processor checks the ES flag in the x87 FPU status word for pending floating-point exceptions. If floating-point exceptions are pending, the x87 FPU makes an implicit call (traps) to the floating-point software exception handler. The exception handler can then execute recovery procedures for selected or all floating-point exceptions.

Synchronization problems occur in the time between the moment when the exception is signaled and when it is actually handled. Because of concurrent execution, integer or system instructions can be executed during this time. It is thus possible for the source or destination operands for a floating-point instruction that faulted to be overwritten in memory, making it impossible for the exception handler to analyze or recover from the exception.

To solve this problem, an exception synchronizing instruction (either a floating-point instruction or a WAIT/FWAIT instruction) can be placed immediately after any floating-point instruction that might present a situation where state information pertaining to a floating-point exception might be lost or corrupted. Floating-point instructions that store data in memory are prime candidates for synchronization. For example, the following three lines of code have the potential for exception synchronization problems:

```
FILD COUNT          ;Floating-point instruction
INC COUNT           ;Integer instruction
FSQRT               ;Subsequent floating-point instruction
```

In this example, the INC instruction modifies the source operand of the floating-point instruction, FILD. If an exception is signaled during the execution of the FILD instruction, the INC instruction would be allowed to overwrite the value stored in the COUNT memory location before the floating-point exception handler is called. With the COUNT variable modified, the floating-point exception handler would not be able to recover from the error.

Rearranging the instructions, as follows, so that the FSQRT instruction follows the FILD instruction, synchronizes floating-point exception handling and eliminates the possibility of the COUNT variable being overwritten before the floating-point exception handler is invoked.

```
FILD COUNT          ;Floating-point instruction
FSQRT               ;Subsequent floating-point instruction synchronizes
                    ;any exceptions generated by the FILD instruction.
INC COUNT           ;Integer instruction
```

The FSQRT instruction does not require any synchronization, because the results of this instruction are stored in the x87 FPU data registers and will remain there, undisturbed, until the next floating-point or WAIT/FWAIT instruction is executed. To absolutely ensure that any exceptions emanating from the FSQRT instruction are handled (for example, prior to a procedure call), a WAIT instruction can be placed directly after the FSQRT instruction.

Note that some floating-point instructions (non-waiting instructions) do not check for pending unmasked exceptions (see Section 8.3.11, “x87 FPU Control Instructions”). They include the FNINIT, FNSTENV, FNSAVE, FNSTSW, FNSTCW, and FNCLEX instructions. When an FNINIT, FNSTENV, FNSAVE, or FNCLEX instruction is executed, all pending exceptions are essentially lost (either the x87 FPU status register is cleared or all exceptions are masked). The FNSTSW and FNSTCW instructions do not check for pending interrupts, but they do not modify the x87 FPU status and control registers. A subsequent “waiting” floating-point instruction can then handle any pending exceptions.

8.7 HANDLING X87 FPU EXCEPTIONS IN SOFTWARE

The x87 FPU in Pentium and later IA-32 processors provides two different modes of operation for invoking a software exception handler for floating-point exceptions: native mode and MS-DOS compatibility mode. The mode of operation is selected by CR0.NE[bit 5]. (See Chapter 2, "System Architecture Overview," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A,* for more information about the NE flag.)

8.7.1 Native Mode

The native mode for handling floating-point exceptions is selected by setting CR0.NE[bit 5] to 1. In this mode, if the x87 FPU detects an exception condition while executing a floating-point instruction and the exception is unmasked (the mask bit for the exception is cleared), the x87 FPU sets the flag for the exception and the ES flag in the x87 FPU status word. It then invokes the software exception handler through the floating-point-error exception (#MF, exception vector 16), immediately before execution of any of the following instructions in the processor's instruction stream:

- The next floating-point instruction, unless it is one of the non-waiting instructions (FNINIT, FNCLEX, FNSTSW, FNSTCW, FNSTENV, and FNSAVE).
- The next WAIT/FWAIT instruction.
- The next MMX instruction.

If the next floating-point instruction in the instruction stream is a non-waiting instruction, the x87 FPU executes the instruction without invoking the software exception handler.

8.7.2 MS-DOS* Compatibility Sub-mode

If CR0.NE[bit 5] is 0, the MS-DOS compatibility mode for handling floating-point exceptions is selected. In this mode, the software exception handler for floating-point exceptions is invoked externally using the processor's FERR#, INTR, and IGNNE# pins. This method of reporting floating-point errors and invoking an exception handler is provided to support the floating-point exception handling mechanism used in PC systems that are running the MS-DOS or Windows* 95 operating system.

Using FERR# and IGNNE# to handle floating-point exception is deprecated by modern operating systems, this approach also limits newer processors to operate with one logical processor active.

The MS-DOS compatibility mode is typically used as follows to invoke the floating-point exception handler:

1. If the x87 FPU detects an unmasked floating-point exception, it sets the flag for the exception and the ES flag in the x87 FPU status word.
2. If the IGNNE# pin is deasserted, the x87 FPU then asserts the FERR# pin either immediately, or else delayed (deferred) until just before the execution of the next waiting floating-point instruction or MMX instruction. Whether the FERR# pin is asserted immediately or delayed depends on the type of processor, the instruction, and the type of exception.
3. If a preceding floating-point instruction has set the exception flag for an unmasked x87 FPU exception, the processor freezes just before executing the **next** WAIT instruction, waiting floating-point instruction, or MMX instruction. Whether the FERR# pin was asserted at the preceding floating-point instruction or is just now being asserted, the freezing of the processor assures that the x87 FPU exception handler will be invoked before the new floating-point (or MMX) instruction gets executed.
4. The FERR# pin is connected through external hardware to IRQ13 of a cascaded, programmable interrupt controller (PIC). When the FERR# pin is asserted, the PIC is programmed to generate an interrupt 75H.
5. The PIC asserts the INTR pin on the processor to signal the interrupt 75H.
6. The BIOS for the PC system handles the interrupt 75H by branching to the interrupt 02H (NMI) interrupt handler.
7. The interrupt 02H handler determines if the interrupt is the result of an NMI interrupt or a floating-point exception.

8. If a floating-point exception is detected, the interrupt 02H handler branches to the floating-point exception handler.

If the IGNNE# pin is asserted, the processor ignores floating-point error conditions. This pin is provided to inhibit floating-point exceptions from being generated while the floating-point exception handler is servicing a previously signaled floating-point exception.

Appendix D, "Guidelines for Writing SIMD Floating-Point Exception Handlers," describes the MS-DOS compatibility mode in much greater detail. This mode is somewhat more complicated in the Intel486 and Pentium processor implementations, as described in Appendix D.

8.7.3 Handling x87 FPU Exceptions in Software

Section 4.9.3, "Typical Actions of a Floating-Point Exception Handler," shows actions that may be carried out by a floating-point exception handler. The state of the x87 FPU can be saved with the FSTENV/FNSTENV or FSAVE/FNSAVE instructions (see Section 8.1.10, "Saving the x87 FPU's State with FSTENV/FNSTENV and FSAVE/FNSAVE").

If the faulting floating-point instruction is followed by one or more non-floating-point instructions, it may not be useful to re-execute the faulting instruction. See Section 8.6, "x87 FPU Exception Synchronization," for more information on synchronizing floating-point exceptions.

In cases where the handler needs to restart program execution with the faulting instruction, the IRET instruction cannot be used directly. The reason for this is that because the exception is not generated until the next floating-point or WAIT/FWAIT instruction following the faulting floating-point instruction, the return instruction pointer on the stack may not point to the faulting instruction. To restart program execution at the faulting instruction, the exception handler must obtain a pointer to the instruction from the saved x87 FPU state information, load it into the return instruction pointer location on the stack, and then execute the IRET instruction.

5. Updates to Chapter 13, Volume 1

Change bars and green text show changes to Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

Changes to this chapter: Added Architectural LBR information throughout the chapter as necessary.

CHAPTER 13
MANAGING STATE USING THE XSAVE FEATURE SET

The XSAVE feature set extends the functionality of the FXSAVE and FXRSTOR instructions (see Section 10.5, "FXSAVE and FXRSTOR Instructions") by supporting the saving and restoring of processor state in addition to the x87 execution environment (**x87 state**) and the registers used by the streaming SIMD extensions (**SSE state**).

The **XSAVE feature set** comprises eight instructions. XGETBV and XSETBV allow software to read and write the extended control register XCR0, which controls the operation of the XSAVE feature set. XSAVE, XSAVEOPT, XSAVEC, and XSAVES are four instructions that save processor state to memory; XRSTOR and XRSTORS are corresponding instructions that load processor state from memory. XGETBV, XSAVE, XSAVEOPT, XSAVEC, and XRSTOR can be executed at any privilege level; XSETBV, XSAVES, and XRSTORS can be executed only if CPL = 0. In addition to XCR0, the XSAVES and XRSTORS instructions are controlled also by the IA32_XSS MSR (index DA0H).

The XSAVE feature set organizes the state that manages into **state components**. Operation of the instructions is based on **state-component bitmaps** that have the same format as XCR0 and as the IA32_XSS MSR: each bit corresponds to a state component. Section 13.1 discusses these state components and bitmaps in more detail.

Section 13.2 describes how the processor enumerates support for the XSAVE feature set and for **XSAVE-enabled features** (those features that require use of the XSAVE feature set for their enabling). Section 13.3 explains how software can enable the XSAVE feature set and XSAVE-enabled features.

The XSAVE feature set allows saving and loading processor state from a region of memory called an **XSAVE area**. Section 13.4 presents details of the XSAVE area and its organization. Each XSAVE-managed state component is associated with a section of the XSAVE area. Section 13.5 describes in detail each of the XSAVE-managed state components.

Section 13.7 through Section 13.12 describe the operation of XSAVE, XRSTOR, XSAVEOPT, XSAVEC, XSAVES, and XRSTORS, respectively.

13.1 XSAVE-SUPPORTED FEATURES AND STATE-COMPONENT BITMAPS

The XSAVE feature set supports the saving and restoring of **state components**, each of which is a discrete set of processor registers (or parts of registers). In general, each such state component corresponds to a particular CPU feature. Such a feature is **XSAVE-supported**. Some XSAVE-supported features use registers in multiple XSAVE-managed state components.

The XSAVE feature set organizes the state components of the XSAVE-supported features using **state-component bitmaps**. A state-component bitmap comprises 64 bits; each bit in such a bitmap corresponds to a single state component. The following bits are defined in state-component bitmaps:

- Bit 0 corresponds to the state component used for the x87 FPU execution environment (**x87 state**). See Section 13.5.1.
- Bit 1 corresponds to the state component used for registers used by the streaming SIMD extensions (**SSE state**). See Section 13.5.2.
- Bit 2 corresponds to the state component used for the additional register state used by the Intel® Advanced Vector Extensions (**AVX state**). See Section 13.5.3.
- Bits 4:3 correspond to the two state components used for the additional register state used by Intel® Memory Protection Extensions (**MPX state**):
  - State component 3 is used for the 4 128-bit bounds registers BND0–BND3 (**BNDREGS state**).
  - State component 4 is used for the 64-bit user-mode MPX configuration register BNDCFGU and the 64-bit MPX status register BNDSTATUS (**BNDCSR state**).
- Bits 7:5 correspond to the three state components used for the additional register state used by Intel® Advanced Vector Extensions 512 (**AVX-512 state**):
  - State component 5 is used for the 8 64-bit opmask registers k0–k7 (**opmask state**).

  - State component 6 is used for the upper 256 bits of the registers ZMM0–ZMM15. These 16 256-bit values are denoted ZMM0_H–ZMM15_H (**ZMM_Hi256 state**).
  - State component 7 is used for the 16 512-bit registers ZMM16–ZMM31 (**Hi16_ZMM state**).
- Bit 8 corresponds to the state component used for the Intel Processor Trace MSRs (**PT state**).
- Bit 9 corresponds to the state component used for the protection-key feature's register PKRU (**PKRU state**). See Section 13.5.7.
- Bits 12:11 correspond to the two state components used for the additional register state used by Control-Flow Enforcement Technology (**CET state**):
  - State component 11 is used for the 2 MSRs controlling user-mode functionality for CET (**CET_U state**).
  - State component 12 is used for the 3 MSRs containing shadow-stack pointers for privilege levels 0–2 (**CET_S state**).
- Bit 13 corresponds to the state component used for an MSR used to control hardware duty cycling (**HDC state**). See Section 13.5.9.
- Bit 15 corresponds to the state component used for last-branch record configuration (**LBR state**). See Section 13.5.10.
- Bit 16 corresponds to the state component used for an MSR used to control hardware P-states (**HWP state**). See Section 13.5.11.

Bit 10, bit 14, and bits in the range 62:17 are not currently defined in state-component bitmaps and are reserved for future expansion. As individual state components are defined using those bits, additional sub-sections will be updated within Section 13.5 over time. Bit 63 is used for special functionality in some bitmaps and does not correspond to any state component.

The state component corresponding to bit *i* of state-component bitmaps is called **state component *i***. Thus, x87 state is state component 0; SSE state is state component 1; AVX state is state component 2; MPX state comprises state components 3–4; AVX-512 state comprises state components 5–7; PT state is state component 8; PKRU state is state component 9; CET state comprises state components 11–12; HDC state is state component 13; LBR state is state component 15; and HWP state is state component 16.

The XSAVE feature set uses state-component bitmaps in multiple ways. Most of the instructions use an implicit operand (in EDX:EAX), called the **instruction mask**, which is the state-component bitmap that specifies the state components on which the instruction operates.

Some state components are **user state components**, and they can be managed by the entire XSAVE feature set. Other state components are **supervisor state components**, and they can be managed only by XSAVES and XRSTORS. The state components corresponding to bit 9 and to bits in the range 7:0 are user state components; those corresponding to bit 8, to bits in the range 13:11, and to bits 16:15 are supervisor state components.

Extended control register XCR0 contains a state-component bitmap that specifies the user state components that software has enabled the XSAVE feature set to manage. If the bit corresponding to a state component is clear in XCR0, instructions in the XSAVE feature set will not operate on that state component, regardless of the value of the instruction mask.

The IA32_XSS MSR (index DA0H) contains a state-component bitmap that specifies the supervisor state components that software has enabled XSAVES and XRSTORS to manage (XSAVE, XSAVEC, XSAVEOPT, and XRSTOR cannot manage supervisor state components). If the bit corresponding to a state component is clear in the IA32_XSS MSR, XSAVES and XRSTORS will not operate on that state component, regardless of the value of the instruction mask.

Some XSAVE-supported features can be used only if XCR0 has been configured so that the features' state components can be managed by the XSAVE feature set. (This applies only to features with user state components.) Such state components and features are **XSAVE-enabled**. In general, the processor will not modify (or allow modification of) the registers of a state component of an XSAVE-enabled feature if the bit corresponding to that state component is clear in XCR0. (If software clears such a bit in XCR0, the processor preserves the corresponding state component.) If an XSAVE-enabled feature has not been fully enabled in XCR0, execution of any instruction defined for that feature causes an invalid-opcode exception (#UD).

As will be explained in Section 13.3, the XSAVE feature set is enabled only if CR4.OSXSAVE[bit 18] = 1. If CR4.OSXSAVE = 0, the processor treats XSAVE-enabled state features and their state components as if all bits in XCR0 were clear; the state components cannot be modified and the features' instructions cannot be executed.

The state components for x87 state, for SSE state, for PT state, for PKRU state, for CET state, for HDC state, for LBR state, and for HWP state are XSAVE-managed but the corresponding features are not XSAVE-enabled. Processors allow modification of this state, as well as execution of x87 FPU instructions and SSE instructions and use of Intel Processor Trace, protection keys, CET, hardware duty cycling, LBRs, and hardware P-states regardless of the value of CR4.OSXSAVE and XCR0.

13.2 ENUMERATION OF CPU SUPPORT FOR XSAVE INSTRUCTIONS AND XSAVE-SUPPORTED FEATURES

A processor enumerates support for the XSAVE feature set and for features supported by that feature set using the CPUID instruction. The following items provide specific details:

- CPUID.1:ECX.XSAVE[bit 26] enumerates general support for the XSAVE feature set:
  - If this bit is 0, the processor does not support any of the following instructions: XGETBV, XRSTOR, XRSTORS, XSAVE, XSAVEC, XSAVEOPT, XSAVES, and XSETBV; the processor provides no further enumeration through CPUID function 0DH (see below).
  - If this bit is 1, the processor supports the following instructions: XGETBV, XRSTOR, XSAVE, and XSETBV.[1] Further enumeration is provided through CPUID function 0DH.

  CR4.OSXSAVE can be set to 1 if and only if CPUID.1:ECX.XSAVE[bit 26] is enumerated as 1.
- CPUID function 0DH enumerates details of CPU support through a set of sub-functions. Software selects a specific sub-function by the value placed in the ECX register. The following items provide specific details:
  - CPUID function 0DH, sub-function 0.
    - EDX:EAX is a bitmap of all the user state components that can be managed using the XSAVE feature set. A bit can be set in XCR0 if and only if the corresponding bit is set in this bitmap. Every processor that supports the XSAVE feature set will set EAX[0] (x87 state) and EAX[1] (SSE state).

      If EAX[*i*] = 1 (for $1 < i < 32$) or EDX[*i*–32] = 1 (for $32 \le i < 63$), sub-function *i* enumerates details for state component *i* (see below).
    - ECX enumerates the size (in bytes) required by the XSAVE instruction for an XSAVE area containing all the user state components supported by this processor.
    - EBX enumerates the size (in bytes) required by the XSAVE instruction for an XSAVE area containing all the user state components corresponding to bits currently set in XCR0.
  - CPUID function 0DH, sub-function 1.
    - EAX[0] enumerates support for the XSAVEOPT instruction. The instruction is supported if and only if this bit is 1. If EAX[0] = 0, execution of XSAVEOPT causes an invalid-opcode exception (#UD).
    - EAX[1] enumerates support for **compaction extensions** to the XSAVE feature set. The following are supported if this bit is 1:
      - The compacted format of the extended region of XSAVE areas (see Section 13.4.3).
      - The XSAVEC instruction. If EAX[1] = 0, execution of XSAVEC causes a #UD.
      - Execution of the compacted form of XRSTOR (see Section 13.8).
    - EAX[2] enumerates support for execution of XGETBV with ECX = 1. This allows software to determine the state of the init optimization. See Section 13.6.
    - EAX[3] enumerates support for XSAVES, XRSTORS, and the IA32_XSS MSR. If EAX[3] = 0, execution of XSAVES or XRSTORS causes a #UD; an attempt to access the IA32_XSS MSR using RDMSR or WRMSR causes a general-protection exception (#GP). Every processor that supports a supervisor state component sets EAX[3]. Every processor that sets EAX[3] (XSAVES, XRSTORS, IA32_XSS) will also set EAX[1] (the compaction extensions).

1. If CPUID.1:ECX.XSAVE[bit 26] = 1, XGETBV and XSETBV may be executed with ECX = 0 (to read and write XCR0). Any support for execution of these instructions with other values of ECX is enumerated separately.

- EAX[31:4] are reserved.
- EBX enumerates the size (in bytes) required by the XSAVES instruction for an XSAVE area containing all the state components corresponding to bits currently set in XCR0 | IA32_XSS.
- EDX:ECX is a bitmap of all the supervisor state components that can be managed by XSAVES and XRSTORS. A bit can be set in the IA32_XSS MSR if and only if the corresponding bit is set in this bitmap.

NOTE

In summary, the XSAVE feature set supports state component *i* ($0 \le i < 63$) if one of the following is true: (1) $i < 32$ and CPUID.(EAX=0DH,ECX=0):EAX[*i*] = 1; (2) $i \ge 32$ and CPUID.(EAX=0DH,ECX=0):EAX[*i*–32] = 1; (3) $i < 32$ and CPUID.(EAX=0DH,ECX=1):ECX[*i*] = 1; or (4) $i \ge 32$ and CPUID.(EAX=0DH,ECX=1):EDX[*i*–32] = 1. The XSAVE feature set supports user state component *i* if (1) or (2) holds; if (3) or (4) holds, state component *i* is a supervisor state component and support is limited to XSAVES and XRSTORS.

— CPUID function 0DH, sub-function *i* ($i > 1$). This sub-function enumerates details for state component *i*. If the XSAVE feature set supports state component *i* (see note above), the following items provide specific details:

- EAX enumerates the size (in bytes) required for state component *i*.
- If state component *i* is a user state component, EBX enumerates the offset (in bytes, from the base of the XSAVE area) of the section used for state component *i*. (This offset applies only when the standard format for the extended region of the XSAVE area is being used; see Section 13.4.3.)
- If state component *i* is a supervisor state component, EBX returns 0.
- If state component *i* is a user state component, ECX[0] return 0; if state component *i* is a supervisor state component, ECX[0] returns 1.
- The value returned by ECX[1] indicates the alignment of state component *i* when the compacted format of the extended region of an XSAVE area is used (see Section 13.4.3). If ECX[1] returns 0, state component *i* is located immediately following the preceding state component; if ECX[1] returns 1, state component *i* is located on the next 64-byte boundary following the preceding state component.
- ECX[31:2] and EDX return 0.

If the XSAVE feature set does not support state component *i*, sub-function *i* returns 0 in EAX, EBX, ECX, and EDX.

13.3 ENABLING THE XSAVE FEATURE SET AND XSAVE-ENABLED FEATURES

Software enables the XSAVE feature set by setting CR4.OSXSAVE[bit 18] to 1 (e.g., with the MOV to CR4 instruction). If this bit is 0, execution of any of XGETBV, XRSTOR, XRSTORS, XSAVE, XSAVEC, XSAVEOPT, XSAVES, and XSETBV causes an invalid-opcode exception (#UD).

When CR4.OSXSAVE = 1 and CPL = 0, executing the XSETBV instruction with ECX = 0 writes the 64-bit value in EDX:EAX to XCR0 (EAX is written to XCR0[31:0] and EDX to XCR0[63:32]). (Execution of the XSETBV instruction causes a general-protection fault — #GP — if CPL > 0.) The following items provide details regarding individual bits in XCR0:

- XCR0[0] is associated with x87 state (see Section 13.5.1). XCR0[0] is always 1. It has that value coming out of RESET. Executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0 and EAX[0] is 0.
- XCR0[1] is associated with SSE state (see Section 13.5.2). Software can use the XSAVE feature set to manage SSE state only if XCR0[1] = 1. The value of XCR0[1] in no way determines whether software can execute SSE instructions (these instructions can be executed even if XCR0[1] = 0).

  XCR0[1] is 0 coming out of RESET. As noted in Section 13.2, every processor that supports the XSAVE feature set allows software to set XCR0[1].
- XCR0[2] is associated with AVX state (see Section 13.5.3). Software can use the XSAVE feature set to manage AVX state only if XCR0[2] = 1. In addition, software can execute AVX instructions only if CR4.OSXSAVE = XCR0[2] = 1. Otherwise, any execution of an AVX instruction causes an invalid-opcode exception (#UD).

XCR0[2] is 0 coming out of RESET. As noted in Section 13.2, a processor allows software to set XCR0[2] if and only if CPUID.(EAX=0DH,ECX=0):EAX[2] = 1. In addition, executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0 and EAX[2:1] has the value 10b; that is, software cannot enable the XSAVE feature set for AVX state but not for SSE state.

As noted in Section 13.1, the processor will preserve AVX state unmodified if software clears XCR0[2]. However, clearing XCR0[2] while AVX state is not in its initial configuration may cause SSE instructions to incur a power and performance penalty. See Section 13.5.3, "Enable the Use Of XSAVE Feature Set And XSAVE State Components" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for how system software can avoid this penalty.

- XCR0[4:3] are associated with MPX state (see Section 13.5.4). Software can use the XSAVE feature set to manage MPX state only if XCR0[4:3] = 11b. In addition, MPX instructions operate as defined only if CR4.OSXSAVE = 1 and XCR0[4:3] = 11b. Otherwise, execution of an MPX instruction causes no operation (as a NOP instruction); in addition, executions of CALL, RET, JMP, and Jcc do not initialize the bounds registers, and they ignore any F2H (BND) prefix.[1]

  XCR0[4:3] have value 00b coming out of RESET. As noted in Section 13.2, a processor allows software to set XCR0[4:3] to 11b if and only if CPUID.(EAX=0DH,ECX=0):EAX[4:3] = 11b. In addition, executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0, EAX[4:3] is neither 00b nor 11b; that is, software can enable the XSAVE feature set for MPX state only if it does so for both state components.

  As noted in Section 13.1, the processor will preserve MPX state unmodified if software clears XCR0[4:3].

- XCR0[7:5] are associated with AVX-512 state (see Section 13.5.5). Software can use the XSAVE feature set to manage AVX-512 state only if XCR0[7:5] = 111b. In addition, software can execute AVX-512 instructions only if CR4.OSXSAVE = 1 and XCR0[7:5] = 111b. Otherwise, any execution of an AVX-512 instruction causes an invalid-opcode exception (#UD).

  XCR0[7:5] have value 000b coming out of RESET. As noted in Section 13.2, a processor allows software to set XCR0[7:5] to 111b if and only if CPUID.(EAX=0DH,ECX=0):EAX[7:5] = 111b. In addition, executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0, EAX[7:5] is not 000b, and any bit is clear in EAX[2:1] or EAX[7:5]; that is, software can enable the XSAVE feature set for AVX-512 state only if it does so for all three state components, and only if it also does so for AVX state and SSE state. This implies that the value of XCR0[7:5] is always either 000b or 111b.

  As noted in Section 13.1, the processor will preserve AVX-512 state unmodified if software clears XCR0[7:5]. However, clearing XCR0[7:5] while AVX-512 state is not in its initial configuration may cause SSE and AVX instructions to incur a power and performance penalty. See Section 13.5.3, "Enable the Use Of XSAVE Feature Set And XSAVE State Components" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for how system software can avoid this penalty.

- XCR0[9] is associated with PKRU state (see Section 13.5.7). Software can use the XSAVE feature set to manage PKRU state only if XCR0[9] = 1. The value of XCR0[9] in no way determines whether software can use protection keys or execute other instructions that access PKRU state (these instructions can be executed even if XCR0[9] = 0).

  XCR0[9] is 0 coming out of RESET. As noted in Section 13.2, a processor allows software to set XCR0[9] if and only if CPUID.(EAX=0DH,ECX=0):EAX[9] = 1.

- XCR0[63:10] and XCR0[8] are reserved.[2] Executing the XSETBV instruction causes a general-protection fault (#GP) if ECX = 0 and any corresponding bit in EDX:EAX is not 0. These bits in XCR0 are all 0 coming out of RESET.

Software operating with CPL > 0 may need to determine whether the XSAVE feature set and certain XSAVE-enabled features have been enabled. If CPL > 0, execution of the MOV from CR4 instruction causes a general-protection fault (#GP). The following alternative mechanisms allow software to discover the enabling of the XSAVE feature set regardless of CPL:

---

1. Prior to the introduction of MPX, the opcodes defining MPX instructions operated as NOP, and the CALL, RET, JMP, and Jcc instructions ignored any F2H prefix.

2. Bit 8 and bits 13:11 correspond to supervisor state components. Since bits can be set in XCR0 only for user state components, those bits of XCR0 must be 0.

- The value of CR4.OSXSAVE is returned in CPUID.1:ECX.OSXSAVE[bit 27]. If software determines that CPUID.1:ECX.OSXSAVE = 1, the processor supports the XSAVE feature set and the feature set has been enabled in CR4.
- Executing the XGETBV instruction with ECX = 0 returns the value of XCR0 in EDX:EAX. XGETBV can be executed if CR4.OSXSAVE = 1 (if CPUID.1:ECX.OSXSAVE = 1), regardless of CPL.

Thus, software can use the following algorithm to determine the support and enabling for the XSAVE feature set:

1. Use CPUID to discover the value of CPUID.1:ECX.OSXSAVE.
   - — If the bit is 0, either the XSAVE feature set is not supported by the processor or has not been enabled by software. Either way, the XSAVE feature set is not available, nor are XSAVE-enabled features such as AVX.
   - — If the bit is 1, the processor supports the XSAVE feature set — including the XGETBV instruction — and it has been enabled by software. The XSAVE feature set can be used to manage x87 state (because XCR0[0] is always 1). Software requiring more detailed information can go on to the next step.
2. Execute XGETBV with ECX = 0 to discover the value of XCR0. If XCR0[1] = 1, the XSAVE feature set can be used to manage SSE state. If XCR0[2] = 1, the XSAVE feature set can be used to manage AVX state and software can execute AVX instructions. If XCR0[4:3] is 11b, the XSAVE feature set can be used to manage MPX state and software can execute MPX instructions. If XCR0[7:5] is 111b, the XSAVE feature set can be used to manage AVX-512 state and software can execute AVX-512 instructions. If XCR0[9] = 1, the XSAVE feature set can be used to manage PKRU state.

The IA32_XSS MSR (with MSR index DA0H) is zero coming out of RESET. If CR4.OSXSAVE = 1, CPUID.(EAX=0DH,ECX=1):EAX[3] = 1, and CPL = 0, executing the WRMSR instruction with ECX = DA0H writes the 64-bit value in EDX:EAX to the IA32_XSS MSR (EAX is written to IA32_XSS[31:0] and EDX to IA32_XSS[63:32]). The following items provide details regarding individual bits in the IA32_XSS MSR:

- IA32_XSS[8] is associated with PT state (see Section 13.5.6). Software can use XSAVES and XRSTORS to manage PT state only if IA32_XSS[8] = 1. The value of IA32_XSS[8] does not determine whether software can use Intel Processor Trace (the feature can be used even if IA32_XSS[8] = 0).
- IA32_XSS[12:11] are associated with CET state (see Section 13.5.8), IA32_XSS[11] with CET_U state and IA32_XSS[12] with CET_S state. Software can use the XSAVES and XRSTORS to manage CET_U state (respectively, CET_S state) only if IA32_XSS[11] = 1 (respectively, IA32_XSS[12] = 1). The value of IA32_XSS[12:11] does not determine whether software can use CET (the feature can be used even if either of IA32_XSS[12:11] is 0).
- IA32_XSS[13] is associated with HDC state (see Section 13.5.9). Software can use XSAVES and XRSTORS to manage HDC state only if IA32_XSS[13] = 1. The value of IA32_XSS[13] does not determine whether software can use hardware duty cycling (the feature can be used even if IA32_XSS[13] = 0).
- IA32_XSS[15] is associated with LBR state (see Section 13.5.10). Software can use XSAVES and XRSTORS to manage LBR state only if IA32_XSS[15] = 1. The value of IA32_XSS[15] does not determine whether software can use LBRs (the feature can be used even if IA32_XSS[15] = 0).
- IA32_XSS[16] is associated with HWP state (see Section 13.5.11). Software can use XSAVES and XRSTORS to manage HWP state only if IA32_XSS[16] = 1. The value of IA32_XSS[16] does not determine whether software can use hardware P-states (the feature can be used even if IA32_XSS[16] = 0).
- IA32_XSS[63:17], IA32_XSS[14], IA32_XSS[10:9] and IA32_XSS[7:0] are reserved.[1] Executing the WRMSR instruction causes a general-protection fault (#GP) if ECX = DA0H and any corresponding bit in EDX:EAX is not 0. These bits in XCR0 are all 0 coming out of RESET.

The IA32_XSS MSR is 0 coming out of RESET.

There is no mechanism by which software operating with CPL > 0 can discover the value of the IA32_XSS MSR.

1. Bit 9 and bits 7:0 correspond to user state components. Since bits can be set in the IA32_XSS MSR only for supervisor state components, those bits of the MSR must be 0.

13.4 XSAVE AREA

The XSAVE feature set includes instructions that save and restore the XSAVE-managed state components to and from memory: XSAVE, XSAVEOPT, XSAVEC, and XSAVES (for saving); and XRSTOR and XRSTORS (for restoring). The processor organizes the state components in a region of memory called an **XSAVE area**. Each of the save and restore instructions takes a memory operand that specifies the 64-byte aligned base address of the XSAVE area on which it operates.

Every XSAVE area has the following format:

- The **legacy region**. The legacy region of an XSAVE area comprises the 512 bytes starting at the area's base address. It is used to manage the state components for x87 state and SSE state. The legacy region is described in more detail in Section 13.4.1.
- The **XSAVE header**. The XSAVE header of an XSAVE area comprises the 64 bytes starting at an offset of 512 bytes from the area's base address. The XSAVE header is described in more detail in Section 13.4.2.
- The **extended region**. The extended region of an XSAVE area starts at an offset of 576 bytes from the area's base address. It is used to manage the state components other than those for x87 state and SSE state. The extended region is described in more detail in Section 13.4.3. The size of the extended region is determined by which state components the processor supports and which bits have been set in XCR0 and IA32_XSS (see Section 13.3).

13.4.1 Legacy Region of an XSAVE Area

The legacy region of an XSAVE area comprises the 512 bytes starting at the area's base address. It has the same format as the FXSAVE area (see Section 10.5.1). The XSAVE feature set uses the legacy area for x87 state (state component 0) and SSE state (state component 1). Table 13-1 illustrates the format of the first 416 bytes of the legacy region of an XSAVE area.

Table 13-1. Format of the Legacy Region of an XSAVE Area

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
FIP[63:48] or reserved	FCS or FIP[47:32]	FIP[31:0]		FOP	Rsvd.	FTW	FSW	FCW	**0**
MXCSR_MASK		MXCSR		FDP[63:48] or reserved	FDS or FDP[47:32]		FDP[31:0]		**16**
Reserved			ST0/MM0						**32**
Reserved			ST1/MM1						**48**
Reserved			ST2/MM2						**64**
Reserved			ST3/MM3						**80**
Reserved			ST4/MM4						**96**
Reserved			ST5/MM5						**112**
Reserved			ST6/MM6						**128**
Reserved			ST7/MM7						**144**
XMM0									**160**
XMM1									**176**
XMM2									**192**
XMM3									**208**
XMM4									**224**
XMM5									**240**
XMM6									**256**
XMM7									**272**

Table 13-1. Format of the Legacy Region of an XSAVE Area (Contd.)

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
XMM8									**288**
XMM9									**304**
XMM10									**320**
XMM11									**336**
XMM12									**352**
XMM13									**368**
XMM14									**384**
XMM15									**400**

The x87 state component comprises bytes 23:0 and bytes 159:32. The SSE state component comprises bytes 31:24 and bytes 415:160. The XSAVE feature set does not use bytes 511:416; bytes 463:416 are reserved.

Section 13.7 through Section 13.9 provide details of how instructions in the XSAVE feature set use the legacy region of an XSAVE area.

13.4.2 XSAVE Header

The XSAVE header of an XSAVE area comprises the 64 bytes starting at offset 512 from the area's base address:

- Bytes 7:0 of the XSAVE header is a state-component bitmap (see Section 13.1) called **XSTATE_BV**. It identifies the state components in the XSAVE area.
- Bytes 15:8 of the XSAVE header is a state-component bitmap called **XCOMP_BV**. It is used as follows:
  - XCOMP_BV[63] indicates the format of the extended region of the XSAVE area (see Section 13.4.3). If it is clear, the standard format is used. If it is set, the compacted format is used; XCOMP_BV[62:0] provide format specifics as specified in Section 13.4.3.
  - XCOMP_BV[63] determines which form of the XRSTOR instruction is used. If the bit is set, the compacted form is used; otherwise, the standard form is used. See Section 13.8.
  - All bits in XCOMP_BV should be 0 if the processor does not support the compaction extensions to the XSAVE feature set.
- Bytes 63:16 of the XSAVE header are reserved.

Section 13.7 through Section 13.9 provide details of how instructions in the XSAVE feature set use the XSAVE header of an XSAVE area.

13.4.3 Extended Region of an XSAVE Area

The extended region of an XSAVE area starts at byte offset 576 from the area's base address. The size of the extended region is determined by which state components the processor supports and which bits have been set in XCR0 | IA32_XSS (see Section 13.3).

The XSAVE feature set uses the extended area for each state component *i*, where $i \geq 2$. The following state components are currently supported in the extended area: state component 2 contains AVX state; state components 5–7 contain AVX-512 state; and state component 9 contains PKRU state.

The extended region of the an XSAVE area may have one of two formats. The **standard format** is supported by all processors that support the XSAVE feature set; the **compacted format** is supported by those processors that support the compaction extensions to the XSAVE feature set (see Section 13.2). Bit 63 of the XCOMP_BV field in the XSAVE header (see Section 13.4.2) indicates which format is used.

The following items describe the two possible formats of the extended region:

- **Standard format.** Each state component *i* (*i* ≥ 2) is located at the byte offset from the base address of the XSAVE area enumerated in CPUID.(EAX=0DH,ECX=*i*):EBX. (CPUID.(EAX=0DH,ECX=*i*):EAX enumerates the number of bytes required for state component *i*.
- **Compacted format.** Each state component *i* (*i* ≥ 2) is located at a byte offset from the base address of the XSAVE area based on the XCOMP_BV field in the XSAVE header:
  - If XCOMP_BV[*i*] = 0, state component *i* is not in the XSAVE area.
  - If XCOMP_BV[*i*] = 1, state component *i* is located at a byte offset $location_I$ from the base address of the XSAVE area, where $location_I$ is determined by the following items:
    - If XCOMP_BV[*j*] = 0 for every *j*, 2 ≤ *j* < *i*, $location_I$ is 576. (This item applies if *i* is the first bit set in bits 62:2 of the XCOMP_BV; it implies that state component *i* is located at the beginning of the extended region.)
    - Otherwise, let *j*, 2 ≤ *j* < *i*, be the greatest value such that XCOMP_BV[*j*] = 1. Then $location_I$ is determined by the following values: $location_J$; $size_J$, as enumerated in CPUID.(EAX=0DH,ECX=*j*):EAX; and the value of $align_I$, as enumerated in CPUID.(EAX=0DH,ECX=*i*):ECX[1]:
      - If $align_I$ = 0, $location_I$ = $location_J$ + $size_J$. (This item implies that state component *i* is located immediately following the preceding state component whose bit is set in XCOMP_BV.)
      - If $align_I$ = 1, $location_I$ = ceiling($location_J$ + $size_J$, 64). (This item implies that state component *i* is located on the next 64-byte boundary following the preceding state component whose bit is set in XCOMP_BV.)

13.5 XSAVE-MANAGED STATE

The section provides details regarding how the XSAVE feature set interacts with the various XSAVE-managed state components.

Unless otherwise state, the state pertaining to a particular state component is saved beginning at byte 0 of the section of the XSAVE are corresponding to that state component.

13.5.1 x87 State

Instructions in the XSAVE feature set can manage the same state of the x87 FPU execution environment (**x87 state**) that can be managed using the FXSAVE and FXRSTOR instructions. They organize all x87 state as a user state component in the legacy region of the XSAVE area (see Section 13.4.1). This region is illustrated in Table 13-1; the x87 state is listed below, along with details of its interactions with the XSAVE feature set:

- Bytes 1:0, 3:2, 7:6. These are used for the x87 FPU Control Word (FCW), the x87 FPU Status Word (FSW), and the x87 FPU Opcode (FOP), respectively.
- Byte 4 is used for an abridged version of the x87 FPU Tag Word (FTW). The following items describe its usage:
  - For each *j*, 0 ≤ *j* ≤ 7, XSAVE, XSAVEOPT, XSAVEC, and XSAVES save a 0 into bit *j* of byte 4 if x87 FPU data register ST*j* has a empty tag; otherwise, XSAVE, XSAVEOPT, XSAVEC, and XSAVES save a 1 into bit *j* of byte 4.
  - For each *j*, 0 ≤ *j* ≤ 7, XRSTOR and XRSTORS establish the tag value for x87 FPU data register ST*j* as follows. If bit *j* of byte 4 is 0, the tag for ST*j* in the tag register for that data register is marked empty (11B); otherwise, the x87 FPU sets the tag for ST*j* based on the value being loaded into that register (see below).
- Bytes 15:8 are used as follows:
  - If the instruction has no REX prefix, or if REX.W = 0:
    - Bytes 11:8 are used for bits 31:0 of the x87 FPU Instruction Pointer Offset (FIP).
    - If CPUID.(EAX=07H,ECX=0H):EBX[bit 13] = 0, bytes 13:12 are used for x87 FPU Instruction Pointer Selector (FCS). Otherwise, XSAVE, XSAVEOPT, XSAVEC, and XSAVES save these bytes as 0000H, and XRSTOR and XRSTORS ignore them.
    - Bytes 15:14 are not used.

  - If the instruction has a REX prefix with REX.W = 1, bytes 15:8 are used for the full 64 bits of FIP.
- Bytes 23:16 are used as follows:
  - If the instruction has no REX prefix, or if REX.W = 0:
    - Bytes 19:16 are used for bits 31:0 of the x87 FPU Data Pointer Offset (FDP).
    - If CPUID.(EAX=07H,ECX=0H):EBX[bit 13] = 0, bytes 21:20 are used for x87 FPU Data Pointer Selector (FDS). Otherwise, XSAVE, XSAVEOPT, XSAVEC, and XSAVES save these bytes as 0000H; and XRSTOR and XRSTORS ignore them.
    - Bytes 23:22 are not used.
  - If the instruction has a REX prefix with REX.W = 1, bytes 23:16 are used for the full 64 bits of FDP.
- Bytes 31:24 are used for SSE state (see Section 13.5.2).
- Bytes 159:32 are used for the registers ST0–ST7 (MM0–MM7). Each of the 8 register is allocated a 128-bit region, with the low 80 bits used for the register and the upper 48 bits unused.

x87 state is XSAVE-managed but the x87 FPU feature is not XSAVE-enabled. The XSAVE feature set can operate on x87 state only if the feature set is enabled (CR4.OSXSAVE = 1).[1] Software can otherwise use x87 state even if the XSAVE feature set is not enabled.

13.5.2 SSE State

Instructions in the XSAVE feature set can manage the registers used by the streaming SIMD extensions (**SSE state**) just as the FXSAVE and FXRSTOR instructions do. They organize all SSE state as a user state component in the legacy region of the XSAVE area (see Section 13.4.1). This region is illustrated in Table 13-1; the SSE state is listed below, along with details of its interactions with the XSAVE feature set:

- Bytes 23:0 are used for x87 state (see Section 13.5.1).
- Bytes 27:24 are used for the MXCSR register. XRSTOR and XRSTORS generate general-protection faults (#GP) in response to attempts to set any of the reserved bits of the MXCSR register.[2]
- Bytes 31:28 are used for the MXCSR_MASK value. XRSTOR and XRSTORS ignore this field.
- Bytes 159:32 are used for x87 state.
- Bytes 287:160 are used for the registers XMM0–XMM7.
- Bytes 415:288 are used for the registers XMM8–XMM15. These fields are used only in 64-bit mode. Executions of XSAVE, XSAVEOPT, XSAVEC, and XSAVES outside 64-bit mode do not modify these bytes; executions of XRSTOR and XRSTORS outside 64-bit mode do not update XMM8–XMM15. See Section 13.13.

SSE state is XSAVE-managed but the SSE feature is not XSAVE-enabled. The XSAVE feature set can operate on SSE state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage SSE state (XCR0[1] = 1). Software can otherwise use SSE state even if the XSAVE feature set is not enabled or has not been configured to manage SSE state.

13.5.3 AVX State

The register state used by the Intel® Advanced Vector Extensions (AVX) comprises the MXCSR register and 16 256-bit vector registers called YMM0–YMM15. The low 128 bits of each register YMM*i* is identical to the SSE register XMM*i*. Thus, the new state register state added by AVX comprises the upper 128 bits of the registers YMM0–YMM15. These 16 128-bit values are denoted YMM0_H–YMM15_H and are collectively called **AVX state**.

As noted in Section 13.1, the XSAVE feature set manages AVX state as user state component 2. Thus, AVX state is located in the extended region of the XSAVE area (see Section 13.4.3).

---

1. The processor ensures that XCR0[0] is always 1.
2. While MXCSR and MXCSR_MASK are part of SSE state, their treatment by the XSAVE feature set is not the same as that of the XMM registers. See Section 13.7 through Section 13.11 for details.

As noted in Section 13.2, CPUID.(EAX=0DH,ECX=2):EBX enumerates the offset (in bytes, from the base of the XSAVE area) of the section of the extended region of the XSAVE area used for AVX state (when the standard format of the extended region is used). CPUID.(EAX=0DH,ECX=2):EAX enumerates the size (in bytes) required for AVX state.

The XSAVE feature set partitions YMM0_H–YMM15_H in a manner similar to that used for the XMM registers (see Section 13.5.2). Bytes 127:0 of the AVX-state section are used for YMM0_H–YMM7_H. Bytes 255:128 are used for YMM8_H–YMM15_H, but they are used only in 64-bit mode. Executions of XSAVE, XSAVEOPT, XSAVEC, and XSAVES outside 64-bit mode do not modify bytes 255:128; executions of XRSTOR and XRSTORS outside 64-bit mode do not update YMM8_H–YMM15_H. See Section 13.13. In general, bytes 16*i*+15:16*i* are used for YMM*i*_H (for $0 \leq i \leq 15$).

AVX state is XSAVE-managed and the AVX feature is XSAVE-enabled. The XSAVE feature set can operate on AVX state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage AVX state (XCR0[2] = 1). AVX instructions cannot be used unless the XSAVE feature set is enabled and has been configured to manage AVX state.

13.5.4 MPX State

The register state used by the Intel® Memory Protection Extensions (MPX) comprises the 4 128-bit bounds registers BND0–BND3 (**BNDREGS state**); and the 64-bit user-mode configuration register BNDCFGU and the 64-bit MPX status register BNDSTATUS (collectively, **BNDCSR state**). Together, these two user state components compose **MPX state**.

As noted in Section 13.1, the XSAVE feature set manages MPX state as state components 3–4. Thus, MPX state is located in the extended region of the XSAVE area (see Section 13.4.3). The following items detail how these state components are organized in this region:

- **BNDREGS state.**
  As noted in Section 13.2, CPUID.(EAX=0DH,ECX=3):EBX enumerates the offset (in bytes, from the base of the XSAVE area) of the section of the extended region of the XSAVE area used for BNDREGS state (when the standard format of the extended region is used). CPUID.(EAX=0DH,ECX=3):EAX enumerates the size (in bytes) required for BNDREGS state. The BNDREGS section is used for the 4 128-bit bound registers BND0–BND3, with bytes 16*i*+15:16*i* being used for BND*i*.
- **BNDCSR state.**
  As noted in Section 13.2, CPUID.(EAX=0DH,ECX=4):EBX enumerates the offset of the section of the extended region of the XSAVE area used for BNDCSR state (when the standard format of the extended region is used). CPUID.(EAX=0DH,ECX=4):EAX enumerates the size (in bytes) required for BNDCSR state. In the BNDSCR section, bytes 7:0 are used for BNDCFGU and bytes 15:8 are used for BNDSTATUS.

Both components of MPX state are XSAVE-managed and the MPX feature is XSAVE-enabled. The XSAVE feature set can operate on MPX state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage MPX state (XCR0[4:3] = 11b). MPX instructions cannot be used unless the XSAVE feature set is enabled and has been configured to manage MPX state.

13.5.5 AVX-512 State

The register state used by the Intel® Advanced Vector Extensions 512 (AVX-512) comprises the MXCSR register, the 8 64-bit opmask registers k0–k7, and 32 512-bit vector registers called ZMM0–ZMM31. For each *i*, 0 <= *i* <= 15, the low 256 bits of register ZMM*i* is identical to the AVX register YMM*i*. Thus, the new state register state added by AVX comprises the following user state components:

- The opmask registers, collectively called **opmask state**.
- The upper 256 bits of the registers ZMM0–ZMM15. These 16 256-bit values are denoted ZMM0_H–ZMM15_H and are collectively called **ZMM_Hi256 state**.
- The 16 512-bit registers ZMM16–ZMM31, collectively called **Hi16_ZMM state**.

Together, these three state components compose **AVX-512 state**.

As noted in Section 13.1, the XSAVE feature set manages AVX-512 state as state components 5–7. Thus, AVX-512 state is located in the extended region of the XSAVE area (see Section 13.4.3). The following items detail how these state components are organized in this region:

- **Opmask state.**
  As noted in Section 13.2, CPUID.(EAX=0DH,ECX=5):EBX enumerates the offset (in bytes, from the base of the XSAVE area) of the section of the extended region of the XSAVE area used for opmask state (when the standard format of the extended region is used). CPUID.(EAX=0DH,ECX=5):EAX enumerates the size (in bytes) required for opmask state. The opmask section is used for the 8 64-bit opmask registers k0–k7, with bytes 8*i*+7:8*i* being used for k*i.*
- **ZMM_Hi256 state.**
  As noted in Section 13.2, CPUID.(EAX=0DH,ECX=6):EBX enumerates the offset of the section of the extended region of the XSAVE area used for ZMM_Hi256 state (when the standard format of the extended region is used). CPUID.(EAX=0DH,ECX=6):EAX enumerates the size (in bytes) required for ZMM_Hi256 state.

  The XSAVE feature set partitions ZMM0_H–ZMM15_H in a manner similar to that used for the XMM registers (see Section 13.5.2). Bytes 255:0 of the ZMM_Hi256-state section are used for ZMM0_H–ZMM7_H. Bytes 511:256 are used for ZMM8_H–ZMM15_H, but they are used only in 64-bit mode. Executions of XSAVE, XSAVEOPT, XSAVEC, and XSAVES outside 64-bit mode do not modify bytes 511:256; executions of XRSTOR and XRSTORS outside 64-bit mode do not update ZMM8_H–ZMM15_H. See Section 13.13. In general, bytes 32*i*+31:32*i* are used for ZMM*i*_H (for $0 \leq i \leq 15$).
- **Hi16_ZMM state.**
  As noted in Section 13.2, CPUID.(EAX=0DH,ECX=7):EBX enumerates the offset of the section of the extended region of the XSAVE area used for Hi16_ZMM state (when the standard format of the extended region is used). CPUID.(EAX=0DH,ECX=7):EAX enumerates the size (in bytes) required for Hi16_ZMM state.

  The XSAVE feature set accesses Hi16_ZMM state only in 64-bit mode. Executions of XSAVE, XSAVEOPT, XSAVEC, and XSAVES outside 64-bit mode do not modify the Hi16_ZMM section; executions of XRSTOR and XRSTORS outside 64-bit mode do not update ZMM16–ZMM31. See Section 13.13. In general, bytes 64(*i*-16)+63:64(*i*-16) are used for ZMM*i* (for $16 \leq i \leq 31$).

All three components of AVX-512 state are XSAVE-managed and the AVX-512 feature is XSAVE-enabled. The XSAVE feature set can operate on AVX-512 state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage AVX-512 state (XCR0[7:5] = 111b). AVX-512 instructions cannot be used unless the XSAVE feature set is enabled and has been configured to manage AVX-512 state.

13.5.6 PT State

The register state used by Intel Processor Trace (**PT state**) comprises the following 9 MSRs: IA32_RTIT_CTL, IA32_RTIT_OUTPUT_BASE, IA32_RTIT_OUTPUT_MASK_PTRS, IA32_RTIT_STATUS, IA32_RTIT_CR3_MATCH, IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B, IA32_RTIT_ADDR1_A, and IA32_RTIT_ADDR1_B.[1]

As noted in Section 13.1, the XSAVE feature set manages PT state as supervisor state component 8. Thus, PT state is located in the extended region of the XSAVE area (see Section 13.4.3). As noted in Section 13.2, CPUID.(EAX=0DH,ECX=8):EAX enumerates the size (in bytes) required for PT state. The MSRs are each allocated 8 bytes in the state component in the order given above. Thus, IA32_RTIT_CTL is at byte offset 0, IA32_RTIT_OUTPUT_BASE at byte offset 8, etc. Any locations in the state component at or beyond byte offset 72 are reserved.

PT state is XSAVE-managed but Intel Processor Trace is not XSAVE-enabled. The XSAVE feature set can operate on PT state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage PT state (IA32_XSS[8] = 1). Software can otherwise use Intel Processor Trace and access its MSRs (using RDMSR and WRMSR) even if the XSAVE feature set is not enabled or has not been configured to manage PT state.

The following items describe special treatment of PT state by the XSAVES and XRSTORS instructions:

1. These MSRs might not be supported by every processor that supports Intel Processor Trace. Software can use the CPUID instruction to discover which are supported; see Section 32.3.1, "Detection of Intel Processor Trace and Capability Enumeration," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.*

- If XSAVES saves PT state, the instruction clears IA32_RTIT_CTL.TraceEn (bit 0) after saving the value of the IA32_RTIT_CTL MSR and before saving any other PT state. If XSAVES causes a fault or a VM exit, it restores IA32_RTIT_CTL.TraceEn to its original value.
- If XSAVES saves PT state, the instruction saves zeroes in the reserved portions of the state component.
- If XRSTORS would restore (or initialize) PT state and IA32_RTIT_CTL.TraceEn = 1, the instruction causes a general-protection exception (#GP) before modifying PT state.
- If XRSTORS causes an exception or a VM exit, it does so before any modification to IA32_RTIT_CTL.TraceEn (even if it has loaded other PT state).

13.5.7 PKRU State

The register state used by the protection-key feature (**PKRU state**) is the 32-bit PKRU register. As noted in Section 13.1, the XSAVE feature set manages PKRU state as user state component 9. Thus, PKRU state is located in the extended region of the XSAVE area (see Section 13.4.3).

As noted in Section 13.2, CPUID.(EAX=0DH,ECX=9):EBX enumerates the offset (in bytes, from the base of the XSAVE area) of the section of the extended region of the XSAVE area used for PKRU state (when the standard format of the extended region is used). CPUID.(EAX=0DH,ECX=9):EAX enumerates the size (in bytes) required for PKRU state. The XSAVE feature set uses bytes 3:0 of the PK-state section for the PKRU register.

PKRU state is XSAVE-managed but the protection-key feature is not XSAVE-enabled. The XSAVE feature set can operate on PKRU state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage PKRU state (XCR0[9] = 1). Software can otherwise use protection keys and access PKRU state even if the XSAVE feature set is not enabled or has not been configured to manage PKRU state.

The value of the PKRU register determines the access rights for user-mode linear addresses. (See Section 4.6, "Access Rights," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.) The access rights that pertain to an execution of the XRSTOR and XRSTORS instructions are determined by the value of the register before the execution and not by any value that the execution might load into the PKRU register.

13.5.8 CET State

The register state used by Control-Flow Enforcement Technology (CET) comprises the two 64-bit MSRs (IA32_U_CET and IA32_PL3_SSP) that manage CET when CPL = 3 (**CET_U state**); and the three 64-bit MSRs (IA32_PL0_SSP–IA32_PL2_SSP) that manage CET when CPL < 3 (**CET_S state**). Together, these two user state components compose **CET state**.[1]

As noted in Section 13.1, the XSAVE feature set manages CET state as supervisor state components 11–23. Thus, CET state is located in the extended region of the XSAVE area (see Section 13.4.3). The following items detail how these state components are organized in this region:

- **CET_U state.**
  As noted in Section 13.2, CPUID.(EAX=0DH,ECX=11):EAX enumerates the size (in bytes) required for CET_U state. The CET_U section is used for the 64-bit MSRs IA32_U_CET and IA32_PL3_SSP, with bytes 7:0 being used for IA32_U_CET and bytes 15:8 being used for IA32_PL3_SSP.
- **CET_S state.**
  As noted in Section 13.2, CPUID.(EAX=0DH,ECX=12):EAX enumerates the size (in bytes) required for CET_S state. The CET_S section is used for the three 64-bit MSRs IA32_PL0_SSP–IA32_PL2_SSP, with bytes 8*i*+7:8*i* being used for IA32_PL*i*_SSP.

The two components of CET state are XSAVE-managed and CET is not XSAVE-enabled. The XSAVE feature set can operate on CET_U state (respectively, CET_S state) only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage CET_U state (respectively, CET_S state) by setting IA32_XSS[11] (respectively, IA32_XSS[12]). Software can otherwise use CET and access the CET MSRs (using RDMSR and WRMSR) even if the XSAVE feature set is not enabled or has not been configured to manage CET state.

1. The IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs also control CET when CPL < 3. However, they are not managed by the XSAVE feature set and are thus not considered in this chapter.

13.5.9 HDC State

The register state used by hardware duty cycling (**HDC state**) comprises the IA32_PM_CTL1 MSR.

As noted in Section 13.1, the XSAVE feature set manages HDC state as supervisor state component 13. Thus, HDC state is located in the extended region of the XSAVE area (see Section 13.4.3). As noted in Section 13.2, CPUID.(EAX=0DH,ECX=13):EAX enumerates the size (in bytes) required for HDC state. The IA32_PM_CTL1 MSR is allocated 8 bytes at byte offset 0 in the state component.

HDC state is XSAVE-managed but hardware duty cycling is not XSAVE-enabled. The XSAVE feature set can operate on HDC state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage HDC state (IA32_XSS[13] = 1). Software can otherwise use hardware duty cycling and access the IA32_PM_CTL1 MSR (using RDMSR and WRMSR) even if the XSAVE feature set is not enabled or has not been configured to manage HDC state.

13.5.10 LBR State

The register state used by last-branch records (**LBR state**) comprises 101 MSRs organized as follows: IA32_LBR_CTL; IA32_LBR_DEPTH; IA32_LER_FROM_IP; IA32_LER_TO_IP; IA32_LER_INFO; and 32 triples of MSRs, IA32_LBR_*i*_FROM_IP, IA32_LBR_*i*_TO_IP, IA32_LBR_*i*_INFO, for each value of *i*, $0 \le i \le 31$.

As noted in Section 13.1, the XSAVE feature set manages LBR state as supervisor state component 15. Thus, LBR state is located in the extended region of the XSAVE area (see Section 13.4.3). As noted in Section 13.2, CPUID.(EAX=0DH,ECX=15):EAX enumerates the size (in bytes) required for LBR state. The IA32_LBR_CTL MSR is allocated 8 bytes at byte offset 0 in the state component. The remaining MSRs are each allocated 8 bytes in the state component in the order given above. Thus, IA32_LBR_DEPTH is at byte offset 8, … , IA32_LBR_0_FROM_IP at byte offset 40, IA32_LBR_0_TO_IP at byte offset 48, IA32_LBR_0_INFO at byte offset 56, IA32_LBR_1_FROM_IP at byte offset 64, …, and IA32_LBR_31_INFO at byte offset 800. Any locations in the state component at or beyond byte offset 808 are reserved.

LBR state is XSAVE-managed but LBRs are not XSAVE-enabled. The XSAVE feature set can operate on LBR state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage LBR state (IA32_XSS[15] = 1). Software can otherwise use LBRs and access the MSRs (using RDMSR and WRMSR) even if the XSAVE feature set is not enabled or has not been configured to manage LBR state.

The following items describe special treatment of LBR state by the XSAVES and XRSTORS instructions:

- If XSAVES would save LBR state and that state is not in its initial configuration (see Section 13.6), the instruction always saves IA32_LBR_CTL, IA32_LBR_DEPTH, IA32_LER_FROM_IP, IA32_LER_TO_IP, and IA32_LER_INFO. It saves the triples IA32_LBR_*i*_FROM_IP, IA32_LBR_*i*_TO_IP, IA32_LBR_*i*_INFO, for each value of *i*, $0 \le i < D$, where D is the value of IA32_LBR_DEPTH. It will not save the values of the remaining triples, although it may access the corresponding fields in the XSAVE area.
- If XSAVES would save LBR state and that state is in its initial configuration, the instruction does not save any LBR state and will not access that component of the XSAVE area.
- If XRSTORS would initialize LBR state, IA32_LBR_DEPTH is not modified and zero is written to the other MSRs that compose LBR state.
- If XRSTORS would restore LBR state, behavior depends on the current value of IA32_LBR_DEPTH and the value of corresponding field in the XSAVE area:
  - If the current value of IA32_LBR_DEPTH equals the value of corresponding field in the XSAVE area, the instruction restores IA32_LBR_CTL, IA32_LER_FROM_IP, IA32_LER_TO_IP, IA32_LER_INFO, and the triples IA32_LBR_*i*_FROM_IP, IA32_LBR_*i*_TO_IP, IA32_LBR_*i*_INFO, for each value of *i*, $0 \le i < D$, where D is the value of IA32_LBR_DEPTH. It will not restore the values of the remaining triples, although it may access the corresponding fields in the XSAVE area.
  - If the IA32_LBR_DEPTH field in the XSAVE area sets any reserved bits, the instruction causes a general-protection exception (#GP).
  - If neither of the previous items apply, the instruction restores IA32_LBR_CTL, IA32_LER_FROM_IP, IA32_LER_TO_IP, and IA32_LER_INFO, but it writes zero to the triples IA32_LBR_*i*_FROM_IP, IA32_LBR_*i*_TO_IP, IA32_LBR_*i*_INFO, for each value of *i*, $0 \le i \le 31$. Such an execution does not modify XINUSE[15] (see Section 13.6 and Section 13.12).

13.5.11 HWP State

The register state used by hardware P-states (**HWP state**) comprises the IA32_HWP_REQUEST MSR.

As noted in Section 13.1, the XSAVE feature set manages HWP state as supervisor state component 16. Thus, HWP state is located in the extended region of the XSAVE area (see Section 13.4.3). As noted in Section 13.2, CPUID.(EAX=0DH,ECX=16):EAX enumerates the size (in bytes) required for HWP state. The IA32_HWP_REQUEST MSR is allocated 8 bytes at byte offset 0 in the state component.

HWP state is XSAVE-managed but the hardware P-states feature is not XSAVE-enabled. The XSAVE feature set can operate on HWP state only if the feature set is enabled (CR4.OSXSAVE = 1) and has been configured to manage HWP state (IA32_XSS[16] = 1). Software can otherwise use hardware P-states and access the IA32_HWP_REQUEST MSR (using RDMSR and WRMSR) even if the XSAVE feature set is not enabled or has not been configured to manage HWP state.

13.6 PROCESSOR TRACKING OF XSAVE-MANAGED STATE

The XSAVEOPT, XSAVEC, and XSAVES instructions use two optimizations to reduce the amount of data that they write to memory. They avoid writing data for any state component known to be in its initial configuration (the **init optimization**). In addition, if either XSAVEOPT or XSAVES is using the same XSAVE area as that used by the most recent execution of XRSTOR or XRSTORS, it may avoid writing data for any state component whose configuration is known not to have been modified since then (the **modified optimization**). (XSAVE does not use these optimizations, and XSAVEC does not use the modified optimization.) The operation of XSAVEOPT, XSAVEC, and XSAVES are described in more detail in Section 13.9 through Section 13.11.

A processor can support the init and modified optimizations with special hardware that tracks the state components that might benefit from those optimizations. Other implementations might not include such hardware; such a processor would always consider each such state component as not in its initial configuration and as modified since the last execution of XRSTOR or XRSTORS.

The following notation describes the state of the init and modified optimizations:

- XINUSE denotes the state-component bitmap corresponding to the init optimization. If XINUSE[*i*] = 0, state component *i* is known to be in its initial configuration; otherwise XINUSE[*i*] = 1. It is possible for XINUSE[*i*] to be 1 even when state component *i* is in its initial configuration. On a processor that does not support the init optimization, XINUSE[*i*] is always 1 for every value of *i*.

  Executing XGETBV with ECX = 1 returns in EDX:EAX the logical-AND of XCR0 and the current value of the XINUSE state-component bitmap. Such an execution of XGETBV always sets EAX[1] to 1 if XCR0[1] = 1 and MXCSR does not have its RESET value of 1F80H. Section 13.2 explains how software can determine whether a processor supports this use of XGETBV.
- XMODIFIED denotes the state-component bitmap corresponding to the modified optimization. If XMODIFIED[*i*] = 0, state component *i* is known not to have been modified since the most recent execution of XRSTOR or XRSTORS; otherwise XMODIFIED[*i*] = 1. It is possible for XMODIFIED[*i*] to be 1 even when state component *i* has not been modified since the most recent execution of XRSTOR or XRSTORS. On a processor that does not support the modified optimization, XMODIFIED[*i*] is always 1 for every value of *i*.

A processor that implements the modified optimization saves information about the most recent execution of XRSTOR or XRSTORS in a quantity called **XRSTOR_INFO**, a 4-tuple containing the following: (1) the CPL; (2) whether the logical processor was in VMX non-root operation; (3) the linear address of the XSAVE area; and (4) the XCOMP_BV field in the XSAVE area. An execution of XSAVEOPT or XSAVES uses the modified optimization only if that execution corresponds to XRSTOR_INFO on these four parameters.

This mechanism implies that, depending on details of the operating system, the processor might determine that an execution of XSAVEOPT by one user application corresponds to an earlier execution of XRSTOR by a different application. For this reason, Intel recommends the application software not use the XSAVEOPT instruction.

The following items specify the initial configuration each state component (for the purposes of defining the XINUSE bitmap):

- **x87 state.** x87 state is in its initial configuration if the following all hold: FCW is 037FH; FSW is 0000H; FTW is FFFFH; FCS and FDS are each 0000H; FIP and FDP are each 00000000_00000000H; each of ST0–ST7 is 0000_00000000_00000000H.

- **SSE state.** In 64-bit mode, SSE state is in its initial configuration if each of XMM0–XMM15 is 0. Outside 64-bit mode, SSE state is in its initial configuration if each of XMM0–XMM7 is 0. XINUSE[1] pertains only to the state of the XMM registers and not to MXCSR. An execution of XRSTOR or XRSTORS outside 64-bit mode does not update XMM8–XMM15. (See Section 13.13.)
- **AVX state.** In 64-bit mode, AVX state is in its initial configuration if each of YMM0_H–YMM15_H is 0. Outside 64-bit mode, AVX state is in its initial configuration if each of YMM0_H–YMM7_H is 0. An execution of XRSTOR or XRSTORS outside 64-bit mode does not update YMM8_H–YMM15_H. (See Section 13.13.)
- **BNDREGS state.** BNDREGS state is in its initial configuration if the value of each of BND0–BND3 is 0.
- **BNDCSR state.** BNDCSR state is in its initial configuration if BNDCFGU and BNDCSR each has value 0.
- **Opmask state.** Opmask state is in its initial configuration if each of the opmask registers k0–k7 is 0.
- **ZMM_Hi256 state.** In 64-bit mode, ZMM_Hi256 state is in its initial configuration if each of ZMM0_H–ZMM15_H is 0. Outside 64-bit mode, ZMM_Hi256 state is in its initial configuration if each of ZMM0_H–ZMM7_H is 0. An execution of XRSTOR or XRSTORS outside 64-bit mode does not update ZMM8_H–ZMM15_H. (See Section 13.13.)
- **Hi16_ZMM state.** In 64-bit mode, Hi16_ZMM state is in its initial configuration if each of ZMM16–ZMM31 is 0. Outside 64-bit mode, Hi16_ZMM state is always in its initial configuration. An execution of XRSTOR or XRSTORS outside 64-bit mode does not update ZMM31–ZMM31. (See Section 13.13.)
- **PT state.** PT state is in its initial configuration if each of the 9 MSRs is 0.
- **PKRU state.** PKRU state is in its initial configuration if the value of the PKRU is 0.
- **PT state.** PT state is in its initial configuration if each of the 9 MSRs is 0.
- **CET_U state.** CET_U state is in its initial configuration if both of the MSRs are 0.
- **CET_S state.** CET_S state is in its initial configuration if each of the three MSRs is 0.
- **HDC state.** HDC state is in its initial configuration if the value of the IA32_PM_CTL1 MSR is 1.
- **LBR state.** LBR state is in its initial configuration if the value of each of the MSRs is 0, with the exception of IA32_LBR_DEPTH. XINUSE[15] does not pertain to IA32_LBR_DEPTH.
- **HWP state.** HWP state is in its initial configuration if the value of the IA32_HWP_REQUEST MSR is 8000FF01H.

13.7 OPERATION OF XSAVE

The XSAVE instruction takes a single memory operand, which is an XSAVE area. In addition, the register pair EDX:EAX is an implicit operand used as a state-component bitmap (see Section 13.1) called the **instruction mask**. The logical-AND of XCR0 and the instruction mask is the **requested-feature bitmap** (**RFBM**) of the user state components to be saved.

The following conditions cause execution of the XSAVE instruction to generate a fault:

- If the XSAVE feature set is not enabled (CR4.OSXSAVE = 0), an invalid-opcode exception (#UD) occurs.
- If CR0.TS[bit 3] is 1, a device-not-available exception (#NM) occurs.
- If the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.[1]

If none of these conditions cause a fault, execution of XSAVE reads the XSTATE_BV field of the XSAVE header (see Section 13.4.2) and writes it back to memory, setting XSTATE_BV[i] ($0 \le i \le 63$) as follows:

- If RFBM[i] = 0, XSTATE_BV[i] is not changed.
- If RFBM[i] = 1, XSTATE_BV[i] is set to the value of XINUSE[i]. Section 13.6 defines XINUSE to describe the processor init optimization and specifies the initial configuration of each state component. The nature of that optimization implies the following:
  - If state component i is in its initial configuration, XINUSE[i] may be either 0 or 1, and XSTATE_BV[i] may be written with either 0 or 1.

1. If CR0.AM = 1, CPL = 3, and EFLAGS.AC =1, an alignment-check exception (#AC) may occur instead of #GP.

XINUSE[1] pertains only to the state of the XMM registers and not to MXCSR. Thus, XSTATE_BV[1] may be written with 0 even if MXCSR does not have its RESET value of 1F80H.

— If state component *i* is not in its initial configuration, XINUSE[*i*] = 1 and XSTATE_BV[*i*] is written with 1.

(As explained in Section 13.6, the initial configurations of some state components may depend on whether the processor is in 64-bit mode.)

The XSAVE instruction does not write any part of the XSAVE header other than the XSTATE_BV field; in particular, it does **not** write to the XCOMP_BV field.

Execution of XSAVE saves into the XSAVE area those state components corresponding to bits that are set in RFBM. State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component *i*, $2 \le i \le 62$, is located in the extended region; the XSAVE instruction always uses the standard format for the extended region (see Section 13.4.3).

The MXCSR register and MXCSR_MASK are part of SSE state (see Section 13.5.2) and are thus associated with RFBM[1]. However, the XSAVE instruction also saves these values when RFBM[2] = 1 (even if RFBM[1] = 0).

See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes. See Section 13.13 for details regarding faults caused by memory accesses.

13.8 OPERATION OF XRSTOR

The XRSTOR instruction takes a single memory operand, which is an XSAVE area. In addition, the register pair EDX:EAX is an implicit operand used as a state-component bitmap (see Section 13.1) called the **instruction mask**. The logical-AND of XCR0 and the instruction mask is the **requested-feature bitmap** (**RFBM**) of the user state components to be restored.

The following conditions cause execution of the XRSTOR instruction to generate a fault:

- If the XSAVE feature set is not enabled (CR4.OSXSAVE = 0), an invalid-opcode exception (#UD) occurs.
- If CR0.TS[bit 3] is 1, a device-not-available exception (#NM) occurs.
- If the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.[1]

After checking for these faults, the XRSTOR instruction reads the XCOMP_BV field in the XSAVE area's XSAVE header (see Section 13.4.2). If XCOMP_BV[63] = 0, the **standard form of XRSTOR** is executed (see Section 13.8.1); otherwise, the **compacted form of XRSTOR** is executed (see Section 13.8.2).[2]

See Section 13.2 for details of how to determine whether the compacted form of XRSTOR is supported.

13.8.1 Standard Form of XRSTOR

The standard from of XRSTOR performs additional fault checking. Either of the following conditions causes a general-protection exception (#GP):

- The XSTATE_BV field of the XSAVE header sets a bit that is not set in XCR0.
- Bytes 23:8 of the XSAVE header are not all 0 (this implies that all bits in XCOMP_BV are 0).[3]

If none of these conditions cause a fault, the processor updates each state component *i* for which RFBM[*i*] = 1. XRSTOR updates state component *i* based on the value of bit *i* in the XSTATE_BV field of the XSAVE header:

---

1. If CR0.AM = 1, CPL = 3, and EFLAGS.AC =1, an alignment-check exception (#AC) may occur instead of #GP.
2. If the processor does not support the compacted form of XRSTOR, it may execute the standard form of XRSTOR without first reading the XCOMP_BV field. A processor supports the compacted form of XRSTOR only if it enumerates CPUID.(EAX=0DH,ECX=1):EAX[1] as 1.
3. Bytes 63:24 of the XSAVE header are also reserved. Software should ensure that bytes 63:16 of the XSAVE header are all 0 in any XSAVE area. (Bytes 15:8 should also be 0 if the XSAVE area is to be used on a processor that does not support the compaction extensions to the XSAVE feature set.)

- If XSTATE_BV[*i*] = 0, the state component is set to its initial configuration. Section 13.6 specifies the initial configuration of each state component.

  The initial configuration of state component 1 pertains only to the XMM registers and not to MXCSR. See below for the treatment of MXCSR

- If XSTATE_BV[*i*] = 1, the state component is loaded with data from the XSAVE area. See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes. See Section 13.13 for details regarding faults caused by memory accesses.

  State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component *i*, $2 \le i \le 62$, is located in the extended region; the standard form of XRSTOR uses the standard format for the extended region (see Section 13.4.3).

The MXCSR register is part of state component 1, SSE state (see Section 13.5.2). However, the standard form of XRSTOR loads the MXCSR register from memory whenever the RFBM[1] (SSE) or RFBM[2] (AVX) is set, regardless of the values of XSTATE_BV[1] and XSTATE_BV[2]. The standard form of XRSTOR causes a general-protection exception (#GP) if it would load MXCSR with an illegal value.

13.8.2 Compacted Form of XRSTOR

The compacted from of XRSTOR performs additional fault checking. Any of the following conditions causes a #GP:

- The XCOMP_BV field of the XSAVE header sets a bit in the range 62:0 that is not set in XCR0.
- The XSTATE_BV field of the XSAVE header sets a bit (including bit 63) that is not set in XCOMP_BV.
- Bytes 63:16 of the XSAVE header are not all 0.

If none of these conditions cause a fault, the processor updates each state component *i* for which RFBM[*i*] = 1. XRSTOR updates state component *i* based on the value of bit *i* in the XSTATE_BV field of the XSAVE header:

- If XSTATE_BV[*i*] = 0, the state component is set to its initial configuration. Section 13.6 specifies the initial configuration of each state component.

  If XSTATE_BV[1] = 0, the compacted form XRSTOR initializes MXCSR to 1F80H. (This differs from the standard from of XRSTOR, which loads MXCSR from the XSAVE area whenever either RFBM[1] or RFBM[2] is set.)

  State component *i* is set to its initial configuration as indicated above if RFBM[*i*] = 1 and XSTATE_BV[*i*] = 0 — **even if XCOMP_BV[*i*] = 0**. This is true for all values of *i*, including 0 (x87 state) and 1 (SSE state).

- If XSTATE_BV[*i*] = 1, the state component is loaded with data from the XSAVE area.[1] See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes. See Section 13.13 for details regarding faults caused by memory accesses.

  State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component *i*, $2 \le i \le 62$, is located in the extended region; the compacted form of the XRSTOR instruction uses the compacted format for the extended region (see Section 13.4.3).

The MXCSR register is part of SSE state (see Section 13.5.2) and is thus loaded from memory if RFBM[1] = XSTATE_BV[*i*] = 1. The compacted form of XRSTOR does not consider RFBM[2] (AVX) when determining whether to update MXCSR. (This is a difference from the standard form of XRSTOR.) The compacted form of XRSTOR causes a general-protection exception (#GP) if it would load MXCSR with an illegal value.

13.8.3 XRSTOR and the Init and Modified Optimizations

Execution of the XRSTOR instruction causes the processor to update its tracking for the init and modified optimizations (see Section 13.6). The following items provide details:

- The processor updates its tracking for the init optimization as follows:
  - If RFBM[*i*] = 0, XINUSE[*i*] is not changed.

1. Earlier fault checking ensured that, if the instruction has reached this point in execution and XSTATE_BV[*i*] is 1, then XCOMP_BV[*i*] is also 1.

  - If RFBM[*i*] = 1 and XSTATE_BV[*i*] = 0, state component *i* may be tracked as init; XINUSE[*i*] may be set to 0 or 1. (As noted in Section 13.6, a processor need not implement the init optimization for state component *i*; a processor that does not do so implicitly maintains XINUSE[*i*] = 1 at all times.)
  - If RFBM[*i*] = 1 and XSTATE_BV[*i*] = 1, state component *i* is tracked as not init; XINUSE[*i*] is set to 1.
- The processor updates its tracking for the modified optimization and records information about the XRSTOR execution for future interaction with the XSAVEOPT and XSAVES instructions (see Section 13.9 and Section 13.11) as follows:
  - If RFBM[*i*] = 0, state component *i* is tracked as modified; XMODIFIED[*i*] is set to 1.
  - If RFBM[*i*] = 1, state component *i* may be tracked as unmodified; XMODIFIED[*i*] may be set to 0 or 1. (As noted in Section 13.6, a processor need not implement the modified optimization for state component *i*; a processor that does not do so implicitly maintains XMODIFIED[*i*] = 1 at all times.)
  - XRSTOR_INFO is set to the 4-tuple $\langle w,x,y,z \rangle$, where w is the CPL (0); x is 1 if the logical processor is in VMX non-root operation and 0 otherwise; *y* is the linear address of the XSAVE area; and *z* is XCOMP_BV. In particular, the standard form of XRSTOR always sets *z* to all zeroes, while the compacted form of XRSTORS never does so (because it sets at least bit 63 to 1).

Note that, if RFBM is entirely zero (e.g., because the instruction mask in EDX:EAX is zero), no state components are modified, the XINUSE bitmap is not modified, and all bits are set in the XMODIFIED bitmap. Thus, if EDX:EAX was zero for the most recent execution of XRSTOR, an execution of XSAVEOPT or XSAVES will identify all state components as modified and will thus not use the modified optimization.

13.9 OPERATION OF XSAVEOPT

The operation of XSAVEOPT is similar to that of XSAVE. Unlike XSAVE, XSAVEOPT uses the init optimization (by which it may omit saving state components that are in their initial configuration) and the modified optimization (by which it may omit saving state components that have not been modified since the last execution of XRSTOR); see Section 13.6. See Section 13.2 for details of how to determine whether XSAVEOPT is supported.

The XSAVEOPT instruction takes a single memory operand, which is an XSAVE area. In addition, the register pair EDX:EAX is an implicit operand used as a state-component bitmap (see Section 13.1) called the **instruction mask**. The logical (bitwise) AND of XCR0 and the instruction mask is the **requested-feature bitmap** (**RFBM**) of the user state components to be saved.

The following conditions cause execution of the XSAVEOPT instruction to generate a fault:

- If the XSAVE feature set is not enabled (CR4.OSXSAVE = 0), an invalid-opcode exception (#UD) occurs.
- If CR0.TS[bit 3] is 1, a device-not-available exception (#NM) occurs.
- If the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.[1]

If none of these conditions cause a fault, execution of XSAVEOPT reads the XSTATE_BV field of the XSAVE header (see Section 13.4.2) and writes it back to memory, setting XSTATE_BV[*i*] ($0 \le i \le 63$) as follows:

- If RFBM[*i*] = 0, XSTATE_BV[*i*] is not changed.
- If RFBM[*i*] = 1, XSTATE_BV[*i*] is set to the value of XINUSE[*i*]. Section 13.6 defines XINUSE to describe the processor init optimization and specifies the initial configuration of each state component. The nature of that optimization implies the following:
  - If the state component is in its initial configuration, XINUSE[*i*] may be either 0 or 1, and XSTATE_BV[*i*] may be written with either 0 or 1.

    XINUSE[1] pertains only to the state of the XMM registers and not to MXCSR. Thus, XSTATE_BV[1] may be written with 0 even if MXCSR does not have its RESET value of 1F80H.
  - If the state component is not in its initial configuration, XSTATE_BV[*i*] is written with 1.

  (As explained in Section 13.6, the initial configurations of some state components may depend on whether the processor is in 64-bit mode.)

1. If CR0.AM = 1, CPL = 3, and EFLAGS.AC =1, an alignment-check exception (#AC) may occur instead of #GP.

The XSAVEOPT instruction does not write any part of the XSAVE header other than the XSTATE_BV field; in particular, it does not write to the XCOMP_BV field.

Execution of XSAVEOPT saves into the XSAVE area those state components corresponding to bits that are set in RFBM (subject to the optimizations described below). State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component i, $2 \le i \le 62$, is located in the extended region; the XSAVEOPT instruction always uses the standard format for the extended region (see Section 13.4.3).

See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes. See Section 13.13 for details regarding faults caused by memory accesses.

Execution of XSAVEOPT performs two optimizations that reduce the amount of data written to memory:

- **Init optimization.**
  If XINUSE[i] = 0, state component i is not saved to the XSAVE area (even if RFBM[i] = 1). (See below for exceptions made for MXCSR.)
- **Modified optimization.**
  Each execution of XRSTOR and XRSTORS establishes XRSTOR_INFO as a 4-tuple $\langle w,x,y,z \rangle$ (see Section 13.8.3 and Section 13.12). Execution of XSAVEOPT uses the modified optimization only if the following all hold for the current value of XRSTOR_INFO:
  - w = CPL;
  - x = 1 if and only if the logical processor is in VMX non-root operation;
  - y is the linear address of the XSAVE area being used by XSAVEOPT; and
  - z is 00000000_00000000H. (This last item implies that XSAVEOPT does not use the modified optimization if the last execution of XRSTOR used the compacted form, or if an execution of XRSTORS followed the last execution of XRSTOR.)

  If XSAVEOPT uses the modified optimization and XMODIFIED[i] = 0 (see Section 13.6), state component i is not saved to the XSAVE area.

  (In practice, the benefit of the modified optimization for state component i depends on how the processor is tracking state component i; see Section 13.6. Limitations on the tracking ability may result in state component i being saved even though is in the same configuration that was loaded by the previous execution of XRSTOR.)

  Depending on details of the operating system, an execution of XSAVEOPT by a user application might use the modified optimization when the most recent execution of XRSTOR was by a different application. Because of this, Intel recommends the application software not use the XSAVEOPT instruction.

The MXCSR register and MXCSR_MASK are part of SSE state (see Section 13.5.2) and are thus associated with bit 1 of RFBM. However, the XSAVEOPT instruction also saves these values when RFBM[2] = 1 (even if RFBM[1] = 0). The init and modified optimizations do not apply to the MXCSR register and MXCSR_MASK.

13.10 OPERATION OF XSAVEC

The operation of XSAVEC is similar to that of XSAVE. Two main differences are (1) XSAVEC uses the compacted format for the extended region of the XSAVE area; and (2) XSAVEC uses the init optimization (see Section 13.6). Unlike XSAVEOPT, XSAVEC does not use the modified optimization. See Section 13.2 for details of how to determine whether XSAVEC is supported.

The XSAVEC instruction takes a single memory operand, which is an XSAVE area. In addition, the register pair EDX:EAX is an implicit operand used as a state-component bitmap (see Section 13.1) called the **instruction mask**. The logical (bitwise) AND of XCR0 and the instruction mask is the **requested-feature bitmap** (**RFBM**) of the user state components to be saved.

The following conditions cause execution of the XSAVEC instruction to generate a fault:

- If the XSAVE feature set is not enabled (CR4.OSXSAVE = 0), an invalid-opcode exception (#UD) occurs.
- If CR0.TS[bit 3] is 1, a device-not-available exception (#NM) occurs.
- If the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.[1]

If none of these conditions cause a fault, execution of XSAVEC writes the XSTATE_BV field of the XSAVE header (see Section 13.4.2), setting XSTATE_BV[*i*] ($0 \le i \le 63$) as follows:[1]

- If RFBM[*i*] = 0, XSTATE_BV[*i*] is written as 0.
- If RFBM[*i*] = 1, XSTATE_BV[*i*] is set to the value of XINUSE[*i*] (see below for an exception made for XSTATE_BV[1]). Section 13.6 defines XINUSE to describe the processor init optimization and specifies the initial configuration of each state component. The nature of that optimization implies the following:
  - If state component *i* is in its initial configuration, XSTATE_BV[*i*] may be written with either 0 or 1.
  - If state component *i* is not in its initial configuration, XSTATE_BV[*i*] is written with 1.

  XINUSE[1] pertains only to the state of the XMM registers and not to MXCSR. However, if RFBM[1] = 1 and MXCSR does not have the value 1F80H, XSAVEC writes XSTATE_BV[1] as 1 even if XINUSE[1] = 0.

  (As explained in Section 13.6, the initial configurations of some state components may depend on whether the processor is in 64-bit mode.)

The XSAVEC instructions sets bit 63 of the XCOMP_BV field of the XSAVE header while writing RFBM[62:0] to XCOMP_BV[62:0]. The XSAVEC instruction does not write any part of the XSAVE header other than the XSTATE_BV and XCOMP_BV fields.

Execution of XSAVEC saves into the XSAVE area those state components corresponding to bits that are set in RFBM (subject to the init optimization described below). State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component *i*, $2 \le i \le 62$, is located in the extended region; the XSAVEC instruction always uses the compacted format for the extended region (see Section 13.4.3).

See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes. See Section 13.13 for details regarding faults caused by memory accesses.

Execution of XSAVEC performs the init optimization to reduce the amount of data written to memory. If XINUSE[*i*] = 0, state component *i* is not saved to the XSAVE area (even if RFBM[*i*] = 1). However, if RFBM[1] = 1 and MXCSR does not have the value 1F80H, XSAVEC saves all of state component 1 (SSE — including the XMM registers) even if XINUSE[1] = 0. Unlike the XSAVE instruction, RFBM[2] does not determine whether XSAVEC saves MXCSR and MXCSR_MASK.

13.11 OPERATION OF XSAVES

The operation of XSAVES is similar to that of XSAVEC. The main differences are (1) XSAVES can be executed only if CPL = 0; (2) XSAVES can operate on the state components whose bits are set in XCR0 | IA32_XSS and can thus operate on supervisor state components; and (3) XSAVES uses the modified optimization (see Section 13.6). See Section 13.2 for details of how to determine whether XSAVES is supported.

The XSAVES instruction takes a single memory operand, which is an XSAVE area. In addition, the register pair EDX:EAX is an implicit operand used as a state-component bitmap (see Section 13.1) called the **instruction mask**. EDX:EAX & (XCR0 | IA32_XSS) (the logical AND the instruction mask with the logical OR of XCR0 and IA32_XSS) is the **requested-feature bitmap** (**RFBM**) of the state components to be saved.

The following conditions cause execution of the XSAVES instruction to generate a fault:

- If the XSAVE feature set is not enabled (CR4.OSXSAVE = 0), an invalid-opcode exception (#UD) occurs.
- If CR0.TS[bit 3] is 1, a device-not-available exception (#NM) occurs.
- If CPL > 0 or if the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.

If none of these conditions cause a fault, execution of XSAVES writes the XSTATE_BV field of the XSAVE header (see Section 13.4.2), setting XSTATE_BV[*i*] ($0 \le i \le 63$) as follows:

- If RFBM[*i*] = 0, XSTATE_BV[*i*] is written as 0.

---

1. If CR0.AM = 1, CPL = 3, and EFLAGS.AC =1, an alignment-check exception (#AC) may occur instead of #GP.

1. Unlike the XSAVE and XSAVEOPT instructions, the XSAVEC instruction does **not** read the XSTATE_BV field of the XSAVE header.

- If RFBM[*i*] = 1, XSTATE_BV[*i*] is set to the value of XINUSE[*i*] (see below for an exception made for XSTATE_BV[1]). Section 13.6 defines XINUSE to describe the processor init optimization and specifies the initial configuration of each state component. The nature of that optimization implies the following:

  — If state component *i* is in its initial configuration, XSTATE_BV[*i*] may be written with either 0 or 1.

  — If state component *i* is not in its initial configuration, XSTATE_BV[*i*] is written with 1.

  XINUSE[1] pertains only to the state of the XMM registers and not to MXCSR. However, if RFBM[1] = 1 and MXCSR does not have the value 1F80H, XSAVES writes XSTATE_BV[1] as 1 even if XINUSE[1] = 0.

  (As explained in Section 13.6, the initial configurations of some state components may depend on whether the processor is in 64-bit mode.)

The XSAVES instructions sets bit 63 of the XCOMP_BV field of the XSAVE header while writing RFBM[62:0] to XCOMP_BV[62:0]. The XSAVES instruction does not write any part of the XSAVE header other than the XSTATE_BV and XCOMP_BV fields.

Execution of XSAVES saves into the XSAVE area those state components corresponding to bits that are set in RFBM (subject to the optimizations described below). State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component *i*, $2 \le i \le 62$, is located in the extended region; the XSAVES instruction always uses the compacted format for the extended region (see Section 13.4.3).

See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes; in particular, see Section 13.5.6 and Section 13.5.10 for special treatment by XSAVES of PT state and LBR state, respectively. See Section 13.13 for details regarding faults caused by memory accesses.

Execution of XSAVES performs the init optimization to reduce the amount of data written to memory. If XINUSE[*i*] = 0, state component *i* is not saved to the XSAVE area (even if RFBM[*i*] = 1). However, if RFBM[1] = 1 and MXCSR does not have the value 1F80H, XSAVES saves all of state component 1 (SSE — including the XMM registers) even if XINUSE[1] = 0.

Like XSAVEOPT, XSAVES may perform the modified optimization. Each execution of XRSTOR and XRSTORS establishes XRSTOR_INFO as a 4-tuple $\langle w,x,y,z \rangle$ (see Section 13.8.3 and Section 13.12). Execution of XSAVES uses the modified optimization only if the following all hold:

- *w* = CPL;
- *x* = 1 if and only if the logical processor is in VMX non-root operation;
- *y* is the linear address of the XSAVE area being used by XSAVEOPT; and
- *z*[63] is 1 and *z*[62:0] = RFBM[62:0]. (This last item implies that XSAVES does not use the modified optimization if the last execution of XRSTOR used the standard form and followed the last execution of XRSTORS.)

If XSAVES uses the modified optimization and XMODIFIED[*i*] = 0 (see Section 13.6), state component *i* is not saved to the XSAVE area.

13.12 OPERATION OF XRSTORS

The operation of XRSTORS is similar to that of XRSTOR. Three main differences are (1) XRSTORS can be executed only if CPL = 0; (2) XRSTORS can operate on the state components whose bits are set in XCR0 | IA32_XSS and can thus operate on supervisor state components; and (3) XRSTORS has only a compacted form (no standard form; see Section 13.8). See Section 13.2 for details of how to determine whether XRSTORS is supported.

The XRSTORS instruction takes a single memory operand, which is an XSAVE area. In addition, the register pair EDX:EAX is an implicit operand used as a state-component bitmap (see Section 13.1) called the **instruction mask**. EDX:EAX & (XCR0 | IA32_XSS) (the logical AND the instruction mask with the logical OR of XCR0 and IA32_XSS) is the **requested-feature bitmap** (**RFBM**) of the state components to be restored.

The following conditions cause execution of the XRSTOR instruction to generate a fault:

- If the XSAVE feature set is not enabled (CR4.OSXSAVE = 0), an invalid-opcode exception (#UD) occurs.
- If CR0.TS[bit 3] is 1, a device-not-available exception (#NM) occurs.

- If CPL > 0 or if the address of the XSAVE area is not 64-byte aligned, a general-protection exception (#GP) occurs.

After checking for these faults, the XRSTORS instruction reads the first 64 bytes of the XSAVE header, including the XSTATE_BV and XCOMP_BV fields (see Section 13.4.2). A #GP occurs if any of the following conditions hold for the values read:

- XCOMP_BV[63] = 0.
- XCOMP_BV sets a bit in the range 62:0 that is not set in XCR0 | IA32_XSS.
- XSTATE_BV sets a bit (including bit 63) that is not set in XCOMP_BV.
- Bytes 63:16 of the XSAVE header are not all 0.

If none of these conditions cause a fault, the processor updates each state component *i* for which RFBM[*i*] = 1. XRSTORS updates state component *i* based on the value of bit *i* in the XSTATE_BV field of the XSAVE header:

- If XSTATE_BV[*i*] = 0, the state component is set to its initial configuration. Section 13.6 specifies the initial configuration of each state component. If XSTATE_BV[1] = 0, XRSTORS initializes MXCSR to 1F80H.

  State component *i* is set to its initial configuration as indicated above if RFBM[*i*] = 1 and XSTATE_BV[*i*] = 0 — **even if XCOMP_BV[*i*] = 0**. This is true for all values of *i*, including 0 (x87 state) and 1 (SSE state).
- If XSTATE_BV[*i*] = 1, the state component is loaded with data from the XSAVE area.[1] See Section 13.5 for specifics for each state component and for details regarding mode-specific operation and operation determined by instruction prefixes; in particular, see Section 13.5.6 and Section 13.5.10 for special treatment by XRSTORS of PT state and LBR state, respectively. See Section 13.13 for details regarding faults caused by memory accesses.

  If XRSTORS is restoring a supervisor state component, the instruction causes a general-protection exception (#GP) if it would load any element of that component with an unsupported value (e.g., by setting a reserved bit in an MSR) or if a bit is set in any reserved portion of the state component in the XSAVE area.

  State components 0 and 1 are located in the legacy region of the XSAVE area (see Section 13.4.1). Each state component *i*, $2 \le i \le 62$, is located in the extended region; XRSTORS uses the compacted format for the extended region (see Section 13.4.3).

  The MXCSR register is part of SSE state (see Section 13.5.2) and is thus loaded from memory if RFBM[1] = XSTATE_BV[*i*] = 1. XRSTORS causes a general-protection exception (#GP) if it would load MXCSR with an illegal value.

If an execution of XRSTORS causes an exception or a VM exit during or after restoring a supervisor state component, each element of that state component may have the value it held before the XRSTORS execution, the value loaded from the XSAVE area, or the element's initial value (as defined in Section 13.6). See Section 13.5.6 for some special treatment of PT state for the case in which XRSTORS causes an exception or a VM exit.

Like XRSTOR, execution of XRSTORS causes the processor to update is tracking for the init and modified optimizations (see Section 13.6 and Section 13.8.3). The following items provide details:

- The processor updates its tracking for the init optimization as follows:
  - If RFBM[*i*] = 0, XINUSE[*i*] is not changed.
  - If RFBM[*i*] = 1 and XSTATE_BV[*i*] = 0, state component *i* may be tracked as init; XINUSE[*i*] may be set to 0 or 1.
  - If RFBM[*i*] = 1 and XSTATE_BV[*i*] = 1, state component *i* is tracked as not init; XINUSE[*i*] is set to 1.[2]
- The processor updates its tracking for the modified optimization and records information about the XRSTORS execution for future interaction with the XSAVEOPT and XSAVES instructions as follows:
  - If RFBM[*i*] = 0, state component *i* is tracked as modified; XMODIFIED[*i*] is set to 1.
  - If RFBM[*i*] = 1, state component *i* may be tracked as unmodified; XMODIFIED[*i*] may be set to 0 or 1.

1. Earlier fault checking ensured that, if the instruction has reached this point in execution and XSTATE_BV[*i*] is 1, then XCOMP_BV[*i*] is also 1.
2. For LBR state (state component 15), XRSTORS may leave XINUSE[15] unmodified in certain situations even if RFBM[15] = 1 = XSTATE_BV[15] = 1. See Section 13.5.10.

— XRSTOR_INFO is set to the 4-tuple $\langle w,x,y,z \rangle$, where w is the CPL; x is 1 if the logical processor is in VMX non-root operation and 0 otherwise; y is the linear address of the XSAVE area; and z is XCOMP_BV (this implies that $z[63] = 1$).

Note that, if RFBM is entirely zero (e.g., because the instruction mask in EDX:EAX is zero), no state components are modified, the XINUSE bitmap is not modified, and all bits are set in the XMODIFIED bitmap. Thus, if EDX:EAX was zero for the most recent execution of XRSTORS, an execution of XSAVEOPT or XSAVES will identify all state components as modified and will thus not use the modified optimization.

13.13 MEMORY ACCESSES BY THE XSAVE FEATURE SET

Each instruction in the XSAVE feature set operates on a set of XSAVE-managed state components. The specific set of components on which an instruction operates is determined by the values of XCR0, the IA32_XSS MSR, EDX:EAX, and (for XRSTOR and XRSTORS) the XSAVE header.

Section 13.4 provides the details necessary to determine the location of each state component for any execution of an instruction in the XSAVE feature set. An execution of an instruction in the XSAVE feature set may access any byte of any state component on which that execution operates.

Section 13.5 provides details of the different XSAVE-managed state components. Some portions of some of these components are accessible only in 64-bit mode. Executions of XRSTOR and XRSTORS outside 64-bit mode will not update those portions; executions of XSAVE, XSAVEC, XSAVEOPT, and XSAVES will not modify the corresponding locations in memory.

Despite this fact, any execution of these instructions outside 64-bit mode may access any byte in any state component on which that execution operates — even those at addresses corresponding to registers that are accessible only in 64-bit mode. As result, such an execution may incur a fault due to an attempt to access such an address.

For example, an execution of XSAVE outside 64-bit mode may incur a page fault if paging does not map as read/write the section of the XSAVE area containing state component 7 (Hi16_ZMM state) — despite the fact that state component 7 can be accessed only in 64-bit mode.

6. Updates to Chapter 1, Volume 2A

Change bars and green text show changes to Chapter 1 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A:* Instruction Set Reference, A-L.

Changes to this chapter: Updated section 1.1 "Intel® 64 and IA-32 Processors Covered in this Manual" to add the 12th generation Intel® Core™ processor. Additional minor corrections in chapter.

CHAPTER 1
ABOUT THIS MANUAL

The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*: *Instruction Set Reference* (order numbers 253666, 253667, 326018 and 334569) are part of a set that describes the architecture and programming environment of all Intel 64 and IA-32 architecture processors. Other volumes in this set are:

- The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture* (Order Number 253665).
- The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A, 3B, 3C & 3D: System Programming Guide* (order numbers 253668, 253669, 326019 and 332831).
- The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers* (order number 335592).

The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1,* describes the basic architecture and programming environment of Intel 64 and IA-32 processors. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D*, describe the instruction set of the processor and the opcode structure. These volumes apply to application programmers and to programmers who write operating systems or executives. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A, 3B, 3C & 3D,* describe the operating-system support environment of Intel 64 and IA-32 processors. These volumes target operating-system and BIOS designers. In addition, the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*, addresses the programming environment for classes of software that host operating systems. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4,* describes the model-specific registers of Intel 64 and IA-32 processors.

1.1 INTEL® 64 AND IA-32 PROCESSORS COVERED IN THIS MANUAL

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™2 Duo processor
- Intel® Core™2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series
- Intel® Core™2 Extreme processor X7000 and X6800 series
- Intel® Core™2 Extreme processor QX6000 series
- Intel® Xeon® processor 7100 series

- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel Atom® processor family
- Intel Atom® processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are built from 45 nm and 32 nm processes
- Intel® Core™ i7 processor
- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Core™ i7-3930K processor
- 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series
- Intel® Xeon® processor E3-1200 product family
- Intel® Xeon® processor E5-2400/1400 product family
- Intel® Xeon® processor E5-4600/2600/1600 product family
- 3rd generation Intel® Core™ processors
- Intel® Xeon® processor E3-1200 v2 product family
- Intel® Xeon® processor E5-2400/1400 v2 product families
- Intel® Xeon® processor E5-4600/2600/1600 v2 product families
- Intel® Xeon® processor E7-8800/4800/2800 v2 product families
- 4th generation Intel® Core™ processors
- The Intel® Core™ M processor family
- Intel® Core™ i7-59xx Processor Extreme Edition
- Intel® Core™ i7-49xx Processor Extreme Edition
- Intel® Xeon® processor E3-1200 v3 product family
- Intel® Xeon® processor E5-2600/1600 v3 product families
- 5th generation Intel® Core™ processors
- Intel® Xeon® processor D-1500 product family
- Intel® Xeon® processor E5 v4 family
- Intel Atom® processor X7-Z8000 and X5-Z8000 series
- Intel Atom® processor Z3400 series
- Intel Atom® processor Z3500 series
- 6th generation Intel® Core™ processors
- Intel® Xeon® processor E3-1500m v5 product family
- 7th generation Intel® Core™ processors
- Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series
- Intel® Xeon® Processor Scalable Family
- 8th generation Intel® Core™ processors
- Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series
- Intel® Xeon® E processors
- 9th generation Intel® Core™ processors
- 2nd generation Intel® Xeon® Processor Scalable Family

- 10th generation Intel® Core™ processors
- 11th generation Intel® Core™ processors
- 3rd generation Intel® Xeon® Processor Scalable Family
- 12th generation Intel® Core™ processors

P6 family processors are IA-32 processors based on the P6 family microarchitecture. This includes the Pentium® Pro, Pentium® II, Pentium® III, and Pentium® III Xeon® processors.

The Pentium® 4, Pentium® D, and Pentium® processor Extreme Editions are based on the Intel NetBurst® microarchitecture. Most early Intel® Xeon® processors are based on the Intel NetBurst® microarchitecture. Intel Xeon processor 5000, 7100 series are based on the Intel NetBurst® microarchitecture.

The Intel® Core™ Duo, Intel® Core™ Solo and dual-core Intel® Xeon® processor LV are based on an improved Pentium® M processor microarchitecture.

The Intel® Xeon® processor 3000, 3200, 5100, 5300, 7200, and 7300 series, Intel® Pentium® dual-core, Intel® Core™2 Duo, Intel® Core™2 Quad, and Intel® Core™2 Extreme processors are based on Intel® Core™ microarchitecture.

The Intel® Xeon® processor 5200, 5400, 7400 series, Intel® Core™2 Quad processor Q9000 series, and Intel® Core™2 Extreme processors QX9000, X9000 series, Intel® Core™2 processor E8000 series are based on Enhanced Intel® Core™ microarchitecture.

The Intel Atom® processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are based on the Intel Atom® microarchitecture and supports Intel 64 architecture.

P6 family, Pentium® M, Intel® Core™ Solo, Intel® Core™ Duo processors, dual-core Intel® Xeon® processor LV, and early generations of Pentium 4 and Intel Xeon processors support IA-32 architecture. The Intel® Atom™ processor Z5xx series support IA-32 architecture.

The Intel® Xeon® processor 3000, 3200, 5000, 5100, 5200, 5300, 5400, 7100, 7200, 7300, 7400 series, Intel® Core™2 Duo, Intel® Core™2 Extreme, Intel® Core™2 Quad processors, Pentium® D processors, Pentium® Dual-Core processor, newer generations of Pentium 4 and Intel Xeon processor family support Intel® 64 architecture.

The Intel® Core™ i7 processor and Intel® Xeon® processor 3400, 5500, 7500 series are based on 45 nm Nehalem microarchitecture. Westmere microarchitecture is a 32 nm version of the Nehalem microarchitecture. Intel® Xeon® processor 5600 series, Intel Xeon processor E7 and various Intel Core i7, i5, i3 processors are based on the Westmere microarchitecture. These processors support Intel 64 architecture.

The Intel® Xeon® processor E5 family, Intel® Xeon® processor E3-1200 family, Intel® Xeon® processor E7-8800/4800/2800 product families, Intel® Core™ i7-3930K processor, and 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Sandy Bridge microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E7-8800/4800/2800 v2 product families, Intel® Xeon® processor E3-1200 v2 product family and 3rd generation Intel® Core™ processors are based on the Ivy Bridge microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E5-4600/2600/1600 v2 product families, Intel® Xeon® processor E5-2400/1400 v2 product families and Intel® Core™ i7-49xx Processor Extreme Edition are based on the Ivy Bridge-E microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E3-1200 v3 product family and 4th Generation Intel® Core™ processors are based on the Haswell microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E5-2600/1600 v3 product families and the Intel® Core™ i7-59xx Processor Extreme Edition are based on the Haswell-E microarchitecture and support Intel 64 architecture.

The Intel Atom® processor Z8000 series is based on the Airmont microarchitecture.

The Intel Atom® processor Z3400 series and the Intel Atom® processor Z3500 series are based on the Silvermont microarchitecture.

The Intel® Core™ M processor family, 5th generation Intel® Core™ processors, Intel® Xeon® processor D-1500 product family and the Intel® Xeon® processor E5 v4 family are based on the Broadwell microarchitecture and support Intel 64 architecture.

The Intel® Xeon® Processor Scalable Family, Intel® Xeon® processor E3-1500m v5 product family and 6th generation Intel® Core™ processors are based on the Skylake microarchitecture and support Intel 64 architecture.

The 7th generation Intel® Core™ processors are based on the Kaby Lake microarchitecture and support Intel 64 architecture.

The Intel Atom® processor C series, the Intel Atom® processor X series, the Intel® Pentium® processor J series, the Intel® Celeron® processor J series, and the Intel® Celeron® processor N series are based on the Goldmont microarchitecture.

The Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series is based on the Knights Landing microarchitecture and supports Intel 64 architecture.

The Intel® Pentium® Silver processor series, the Intel® Celeron® processor J series, and the Intel® Celeron® processor N series are based on the Goldmont Plus microarchitecture.

The 8th generation Intel® Core™ processors, 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on the Coffee Lake microarchitecture and support Intel 64 architecture.

The Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series is based on the Knights Mill microarchitecture and supports Intel 64 architecture.

The 2nd generation Intel® Xeon® Processor Scalable Family is based on the Cascade Lake product and supports Intel 64 architecture.

Some 10th generation Intel® Core™ processors are based on the Ice Lake microarchitecture, and some are based on the Comet Lake microarchitecture; both support Intel 64 architecture.

Some 11th generation Intel® Core™ processors are based on the Tiger Lake microarchitecture, and some are based on the Rocket Lake microarchitecture; both support Intel 64 architecture.

Some 3rd generation Intel® Xeon® Processor Scalable Family processors are based on the Cooper Lake product, and some are based on the Ice Lake microarchitecture; both support Intel 64 architecture.

The 12th generation Intel® Core™ processors are based on the Alder Lake performance hybrid architecture and support Intel 64 architecture.

IA-32 architecture is the instruction set architecture and programming environment for Intel's 32-bit microprocessors. Intel® 64 architecture is the instruction set architecture and programming environment which is the superset of Intel’s 32-bit and 64-bit architectures. It is compatible with the IA-32 architecture.

1.2 OVERVIEW OF VOLUME 2A, 2B, 2C AND 2D: INSTRUCTION SET REFERENCE

A description of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B, 2C & 2D* content follows:

Chapter 1 — About This Manual. Gives an overview of all seven volumes of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual*. It also describes the notational conventions in these manuals and lists related Intel® manuals and documentation of interest to programmers and hardware designers.

Chapter 2 — Instruction Format. Describes the machine-level instruction format used for all IA-32 instructions and gives the allowable encodings of prefixes, the operand-identifier byte (ModR/M byte), the addressing-mode specifier byte (SIB byte), and the displacement and immediate bytes.

Chapter 3 — Instruction Set Reference, A-L. Describes Intel 64 and IA-32 instructions in detail, including an algorithmic description of operations, the effect on flags, the effect of operand- and address-size attributes, and the exceptions that may be generated. The instructions are arranged in alphabetical order. General-purpose, x87 FPU, Intel MMX™ technology, SSE/SSE2/SSE3/SSSE3/SSE4 extensions, and system instructions are included.

Chapter 4 — Instruction Set Reference, M-U. Continues the description of Intel 64 and IA-32 instructions started in Chapter 3. It starts *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

Chapter 5 — Instruction Set Reference, V-Z. Continues the description of Intel 64 and IA-32 instructions started in chapters 3 and 4. It provides the balance of the alphabetized list of instructions and starts *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2C*.

Chapter 6 — Safer Mode Extensions Reference. Describes the safer mode extensions (SMX). SMX is intended for a system executive to support launching a measured environment in a platform where the identity of the software controlling the platform hardware can be measured for the purpose of making trust decisions. This chapter starts *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D*.

Chapter 7— Instruction Set Reference Unique to Intel® Xeon Phi™ Processors. Describes the instruction set that is unique to Intel® Xeon Phi™ processors based on the Knights Landing and Knights Mill microarchitectures. The set is not supported in any other Intel processors.

Appendix A — Opcode Map. Gives an opcode map for the IA-32 instruction set.

Appendix B — Instruction Formats and Encodings. Gives the binary encoding of each form of each IA-32 instruction.

Appendix C — Intel® C/C++ Compiler Intrinsics and Functional Equivalents. Lists the Intel® C/C++ compiler intrinsics and their assembly code equivalents for each of the IA-32 MMX and SSE/SSE2/SSE3 instructions.

1.3 NOTATIONAL CONVENTIONS

This manual uses specific notation for data-structure formats, for symbolic representation of instructions, and for hexadecimal and binary numbers. A review of this notation makes the manual easier to read.

1.3.1 Bit and Byte Order

In illustrations of data structures in memory, smaller addresses appear toward the bottom of the figure; addresses increase toward the top. Bit positions are numbered from right to left. The numerical value of a set bit is equal to two raised to the power of the bit position. IA-32 processors are "little endian" machines; this means the bytes of a word are numbered starting from the least significant byte. Figure 1-1 illustrates these conventions.

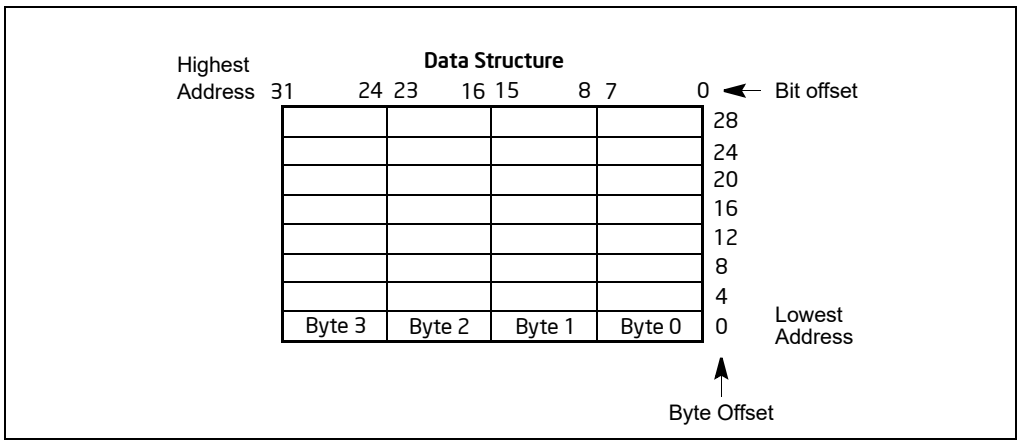


Figure 1-1. Bit and Byte Order

1.3.2 Reserved Bits and Software Compatibility

In many register and memory layout descriptions, certain bits are marked as **reserved**. When bits are marked as reserved, it is essential for compatibility with future processors that software treat these bits as having a future, though unknown, effect. The behavior of reserved bits should be regarded as not only undefined, but unpredictable. Software should follow these guidelines in dealing with reserved bits:

- Do not depend on the states of any reserved bits when testing the values of registers which contain such bits. Mask out the reserved bits before testing.
- Do not depend on the states of any reserved bits when storing to memory or to a register.
- Do not depend on the ability to retain information written into any reserved bits.

- When loading a register, always load the reserved bits with the values indicated in the documentation, if any, or reload them with values previously read from the same register.

NOTE

Avoid any software dependence upon the state of reserved bits in IA-32 registers. Depending upon the values of reserved register bits will make software dependent upon the unspecified manner in which the processor handles these bits. Programs that depend upon reserved values risk incompatibility with future processors.

1.3.3 Instruction Operands

When instructions are represented symbolically, a subset of the IA-32 assembly language is used. In this subset, an instruction has the following format:

```
label: mnemonic argument1, argument2, argument3
```

where:

- A **label** is an identifier which is followed by a colon.
- A **mnemonic** is a reserved name for a class of instruction opcodes which have the same function.
- The operands *argument1*, *argument2*, and *argument3* are optional. There may be from zero to three operands, depending on the opcode. When present, they take the form of either literals or identifiers for data items. Operand identifiers are either reserved names of registers or are assumed to be assigned to data items declared in another part of the program (which may not be shown in the example).

When two operands are present in an arithmetic or logical instruction, the right operand is the source and the left operand is the destination.

For example:

```
LOADREG: MOV EAX, SUBTOTAL
```

In this example, LOADREG is a label, MOV is the mnemonic identifier of an opcode, EAX is the destination operand, and SUBTOTAL is the source operand. Some assembly languages put the source and destination in reverse order.

1.3.4 Hexadecimal and Binary Numbers

Base 16 (hexadecimal) numbers are represented by a string of hexadecimal digits followed by the character H (for example, F82EH). A hexadecimal digit is a character from the following set: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Base 2 (binary) numbers are represented by a string of 1s and 0s, sometimes followed by the character B (for example, 1010B). The "B" designation is only used in situations where confusion as to the type of number might arise.

1.3.5 Segmented Addressing

The processor uses byte addressing. This means memory is organized and accessed as a sequence of bytes. Whether one or more bytes are being accessed, a byte address is used to locate the byte or bytes in memory. The range of memory that can be addressed is called an **address space**.

The processor also supports segmented addressing. This is a form of addressing where a program may have many independent address spaces, called **segments**. For example, a program can keep its code (instructions) and stack in separate segments. Code addresses would always refer to the code space, and stack addresses would always refer to the stack space. The following notation is used to specify a byte address within a segment:

```
Segment-register:Byte-address
```

For example, the following segment address identifies the byte at address FF79H in the segment pointed by the DS register:

```
DS:FF79H
```

The following segment address identifies an instruction address in the code segment. The CS register points to the code segment and the EIP register contains the address of the instruction.

```
CS:EIP
```

1.3.6 Exceptions

An exception is an event that typically occurs when an instruction causes an error. For example, an attempt to divide by zero generates an exception. However, some exceptions, such as breakpoints, occur under other conditions. Some types of exceptions may provide error codes. An error code reports additional information about the error. An example of the notation used to show an exception and error code is shown below:

```
#PF(fault code)
```

This example refers to a page-fault exception under conditions where an error code naming a type of fault is reported. Under some conditions, exceptions which produce error codes may not be able to report an accurate code. In this case, the error code is zero, as shown below for a general-protection exception:

```
#GP(0)
```

1.3.7 A New Syntax for CPUID, CR, and MSR Values

Obtain feature flags, status, and system information by using the CPUID instruction, by checking control register bits, and by reading model-specific registers. We are moving toward a new syntax to represent this information. See Figure 1-2.

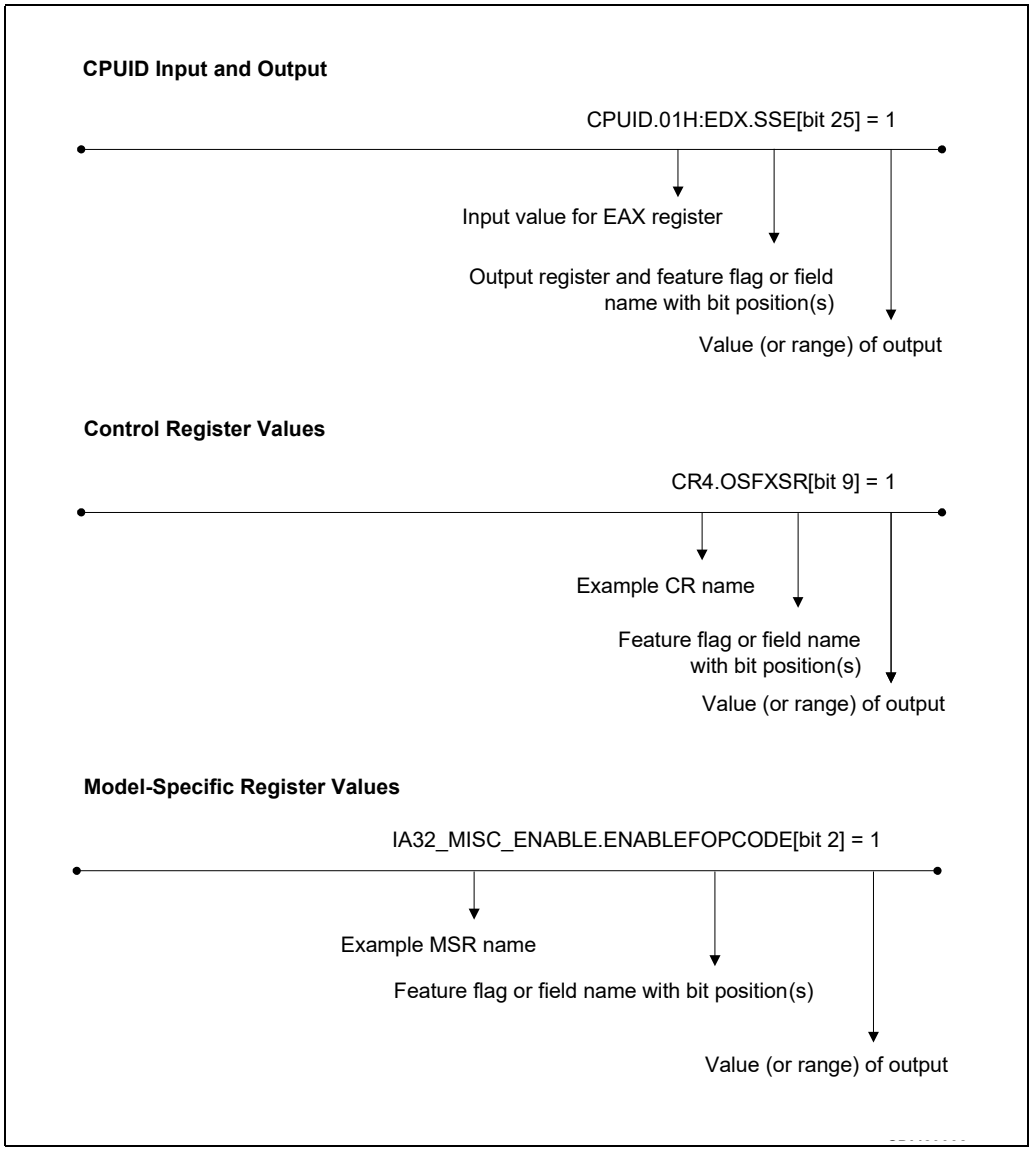


Figure 1-2. Syntax for CPUID, CR, and MSR Data Presentation

1.4 RELATED LITERATURE

Literature related to Intel 64 and IA-32 processors is listed and viewable on-line at:

https://software.intel.com/en-us/articles/intel-sdm

See also:

- The latest security information on Intel® products:
  https://www.intel.com/content/www/us/en/security-center/default.html
- Software developer resources, guidance and insights for security advisories:
  https://software.intel.com/security-software-guidance/
- The data sheet for a particular Intel 64 or IA-32 processor
- The specification update for a particular Intel 64 or IA-32 processor
- Intel® C++ Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/

- Intel® Fortran Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Software Development Tools:
  https://software.intel.com/en-us/intel-sdp-home
- Intel® 64 and IA-32 Architectures Software Developer's Manual (in one, four or ten volumes):
  https://software.intel.com/en-us/articles/intel-sdm
- Intel® 64 and IA-32 Architectures Optimization Reference Manual:
  https://software.intel.com/en-us/articles/intel-sdm#optimization
- Intel® Trusted Execution Technology Measured Launched Environment Programming Guide:

  http://www.intel.com/content/www/us/en/software-developers/intel-txt-software-development-guide.html
- Intel® Software Guard Extensions (Intel® SGX) Information
  https://software.intel.com/en-us/isa-extensions/intel-sgx
- Developing Multi-threaded Applications: A Platform Consistent Approach:
  https://software.intel.com/sites/default/files/article/147714/51534-developing-multithreaded-applications.pdf
- Using Spin-Loops on Intel® Pentium® 4 Processor and Intel® Xeon® Processor:
  https://software.intel.com/sites/default/files/22/30/25602
- Performance Monitoring Unit Sharing Guide
  http://software.intel.com/file/30388

Literature related to select features in future Intel processors are available at:

- Intel® Architecture Instruction Set Extensions Programming Reference
  https://software.intel.com/en-us/isa-extensions

More relevant links are:

- Intel® Developer Zone:

  https://software.intel.com/en-us
- Developer centers:

  http://www.intel.com/content/www/us/en/hardware-developers/developer-centers.html
- Processor support general link:

  http://www.intel.com/support/processors/
- Intel® Hyper-Threading Technology (Intel® HT Technology):

  http://www.intel.com/technology/platform-technology/hyper-threading/index.htm

7. Updates to Chapter 3, Volume 2A

Change bars and green 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: Updated m32 description in Section 3.1.1.3, "Instruction Column in the Opcode Summary Table". Updates to the CPUID, ENTER, and INT n/INTO/INT3/INT1 instructions. Added the HRESET instruction.

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*, and Chapter 5, "Instruction Set Reference, V-Z," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C*.

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	ZO	V/V	NA	Complement carry flag.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
ZO	NA	NA	NA	NA

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.
- **NFx** — 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 or VEX.LZ) or 1 (denoted by VEX.256). 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.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 consider as an implies 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 o f 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 2.6.1 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 is a signed number between –128 and +127 inclusive. 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.
- **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.
- **moffs8, moffs16, moffs32, moffs64** — 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 moffs 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 MODMR.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 disp8*N encoding of the displacement bytes, where N is defined in Table 2-34 and Table 2-35, according to tupletypes. The tupletype 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.

CPUID—CPU Identification

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F A2	CPUID	ZO	Valid	Valid	Returns processor identification and feature information to the EAX, EBX, ECX, and EDX registers, as determined by input entered in EAX (in some cases, ECX as well).

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
ZO	NA	NA	NA	NA

Description

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. This instruction operates the same in non-64-bit modes and 64-bit mode.

CPUID returns processor identification and feature information in the EAX, EBX, ECX, and EDX registers.[1] The instruction's output is dependent on the contents of the EAX register upon execution (in some cases, ECX as well). For example, the following pseudocode loads EAX with 00H and causes CPUID to return a Maximum Return Value and the Vendor Identification String in the appropriate registers:

```
MOV EAX, 00H
CPUID
```

Table 3-8 shows information returned, depending on the initial value loaded into the EAX register.

Two types of information are returned: basic and extended function information. 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. For example, using some Intel processors, the following is true:

```
CPUID.EAX = 05H (* Returns MONITOR/MWAIT leaf. *)
CPUID.EAX = 0AH (* Returns Architectural Performance Monitoring leaf. *)
CPUID.EAX = 0BH (* Returns Extended Topology Enumeration leaf. *)[2]
CPUID.EAX =1FH (* Returns V2 Extended Topology Enumeration leaf. *)[2]
CPUID.EAX = 80000008H (* Returns linear/physical address size data. *)
CPUID.EAX = 8000000AH (* INVALID: Returns same information as CPUID.EAX = 0BH. *)
```

If a value entered for CPUID.EAX is less than or equal to the maximum input value and the leaf is not supported on that processor then 0 is returned in all the registers.

When CPUID returns the highest basic leaf information as a result of an invalid input EAX value, any dependence on input ECX value in the basic leaf is honored.

CPUID can be executed at any privilege level to serialize instruction execution. Serializing instruction execution guarantees that any modifications to flags, registers, and memory for previous instructions are completed before the next instruction is fetched and executed.

See also:

"Serializing Instructions" in Chapter 8, "Multiple-Processor Management," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

"Caching Translation Information" in Chapter 4, "Paging," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

1. On Intel 64 processors, CPUID clears the high 32 bits of the RAX/RBX/RCX/RDX registers in all modes.
2. CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of CPUID leaf 1FH before using leaf 0BH.

Table 3-8. Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
	Basic CPUID Information	
0H	EAX	Maximum Input Value for Basic CPUID Information.
	EBX	"Genu"
	ECX	"ntel"
	EDX	"inel"
01H	EAX	Version Information: Type, Family, Model, and Stepping ID (see Figure 3-6).
	EBX	Bits 07 - 00: Brand Index. Bits 15 - 08: CLFLUSH line size (Value ∗ 8 = cache line size in bytes; used also by CLFLUSHOPT). Bits 23 - 16: Maximum number of addressable IDs for logical processors in this physical package*. Bits 31 - 24: Initial APIC ID**.
	ECX	Feature Information (see Figure 3-7 and Table 3-10).
	EDX	Feature Information (see Figure 3-8 and Table 3-11).
		NOTES: * 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.1.EDX.HTT[bit 28]= 1. ** *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.*
02H	EAX	Cache and TLB Information (see Table 3-12).
	EBX	Cache and TLB Information.
	ECX	Cache and TLB Information.
	EDX	Cache and TLB Information.
03H	EAX	Reserved.
	EBX	Reserved.
	ECX	Bits 00 - 31 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
	EDX	Bits 32 - 63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
		NOTES: Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature.
CPUID leaves above 2 and below 80000000H are visible only when IA32_MISC_ENABLE[bit 22] has its default value of 0.		
	Deterministic Cache Parameters Leaf	
04H		**NOTES:** Leaf 04H output depends on the initial value in ECX.* See also: "INPUT EAX = 04H: Returns Deterministic Cache Parameters for Each Level" on page 245.
	EAX	Bits 04 - 00: Cache Type Field. 0 = Null - No more caches. 1 = Data Cache. 2 = Instruction Cache. 3 = Unified Cache. 4-31 = Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
		Bits 07 - 05: Cache Level (starts at 1). Bit 08: Self Initializing cache level (does not need SW initialization). Bit 09: Fully Associative cache. Bits 13 - 10: Reserved. Bits 25 - 14: Maximum number of addressable IDs for logical processors sharing this cache**, ***. Bits 31 - 26: Maximum number of addressable IDs for processor cores in the physical package**, ****, *****.
	EBX	Bits 11 - 00: L = System Coherency Line Size**. Bits 21 - 12: P = Physical Line partitions**. Bits 31 - 22: W = Ways of associativity**.
	ECX	Bits 31-00: S = Number of Sets**.
	EDX	Bit 00: Write-Back Invalidate/Invalidate. 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. Bit 01: Cache Inclusiveness. 0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels. Bit 02: Complex Cache Indexing. 0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits. Bits 31 - 03: Reserved = 0. **NOTES:** * 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. ** 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. **** 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.
	MONITOR/MWAIT Leaf	
05H	EAX	Bits 15 - 00: Smallest monitor-line size in bytes (default is processor's monitor granularity). Bits 31 - 16: Reserved = 0.
	EBX	Bits 15 - 00: Largest monitor-line size in bytes (default is processor's monitor granularity). Bits 31 - 16: Reserved = 0.
	ECX	Bit 00: Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported. Bit 01: Supports treating interrupts as break-event for MWAIT, even when interrupts disabled. Bits 31 - 02: Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 03 - 00: Number of C0* sub C-states supported using MWAIT. Bits 07 - 04: Number of C1* sub C-states supported using MWAIT. Bits 11 - 08: Number of C2* sub C-states supported using MWAIT. Bits 15 - 12: Number of C3* sub C-states supported using MWAIT. Bits 19 - 16: Number of C4* sub C-states supported using MWAIT. Bits 23 - 20: Number of C5* sub C-states supported using MWAIT. Bits 27 - 24: Number of C6* sub C-states supported using MWAIT. Bits 31 - 28: Number of C7* sub C-states supported using MWAIT. **NOTE:** * The definition of C0 through C7 states for MWAIT extension are processor-specific C-states, not ACPI C-states.
Thermal and Power Management Leaf		
06H	EAX	Bit 00: Digital temperature sensor is supported if set. Bit 01: Intel Turbo Boost Technology available (see description of IA32_MISC_ENABLE[38]). Bit 02: ARAT. APIC-Timer-always-running feature is supported if set. Bit 03: Reserved. Bit 04: PLN. Power limit notification controls are supported if set. Bit 05: ECMD. Clock modulation duty cycle extension is supported if set. Bit 06: PTM. Package thermal management is supported if set. Bit 07: HWP. HWP base registers (IA32_PM_ENABLE[bit 0], IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS) are supported if set. Bit 08: HWP_Notification. IA32_HWP_INTERRUPT MSR is supported if set. Bit 09: HWP_Activity_Window. IA32_HWP_REQUEST[bits 41:32] is supported if set. Bit 10: HWP_Energy_Performance_Preference. IA32_HWP_REQUEST[bits 31:24] is supported if set. Bit 11: HWP_Package_Level_Request. IA32_HWP_REQUEST_PKG MSR is supported if set. Bit 12: Reserved. Bit 13: HDC. HDC base registers IA32_PKG_HDC_CTL, IA32_PM_CTL1, IA32_THREAD_STALL MSRs are supported if set. Bit 14: Intel® Turbo Boost Max Technology 3.0 available. Bit 15: HWP Capabilities. Highest Performance change is supported if set. Bit 16: HWP PECI override is supported if set. Bit 17: Flexible HWP is supported if set. Bit 18: Fast access mode for the IA32_HWP_REQUEST MSR is supported if set. Bit 19: HW_FEEDBACK. 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 are supported if set. Bit 20: Ignoring Idle Logical Processor HWP request is supported if set. Bits 22 - 21: Reserved. Bit 23: Intel® Thread Director supported if set. IA32_HW_FEEDBACK_CHAR and IA32_HW_FEEDBACK_THREAD_CONFIG MSRs are supported if set. Bits 31 - 24: Reserved.
	EBX	Bits 03 - 00: Number of Interrupt Thresholds in Digital Thermal Sensor. Bits 31 - 04: Reserved.
	ECX	Bit 00: Hardware Coordination Feedback Capability (Presence of IA32_MPERF and IA32_APERF). The capability to 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. Bits 02 - 01: Reserved = 0. Bit 03: The processor supports performance-energy bias preference if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H). Bits 07 - 04: Reserved = 0. Bits 15 - 08: 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. Bits 31 - 16: Reserved = 0.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 7-0: Bitmap of supported hardware feedback interface capabilities. 0 = When set to 1, indicates support for performance capability reporting. 1 = When set to 1, indicates support for energy efficiency capability reporting. 2-7 = Reserved Bits 11-8: 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. Bits 31-16: 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. **NOTE:** Bits 0 and 1 will always be set together.
Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)		
07H	Sub-leaf 0 (Input ECX = 0). *	
	EAX	Bits 31 - 00: Reports the maximum input value for supported leaf 7 sub-leaves.
	EBX	Bit 00: FSGSBASE. Supports RDFSBASE/RDGSBASE/WRFSBASE/WRGSBASE if 1. Bit 01: IA32_TSC_ADJUST MSR is supported if 1. Bit 02: SGX. Supports Intel® Software Guard Extensions (Intel® SGX Extensions) if 1. Bit 03: BMI1. Bit 04: HLE. Bit 05: AVX2. Bit 06: FDP_EXCPTN_ONLY. x87 FPU Data Pointer updated only on x87 exceptions if 1. Bit 07: SMEP. Supports Supervisor-Mode Execution Prevention if 1. Bit 08: BMI2. Bit 09: Supports Enhanced REP MOVSB/STOSB if 1. Bit 10: INVPCID. If 1, supports INVPCID instruction for system software that manages process-context identifiers. Bit 11: RTM. Bit 12: RDT-M. Supports Intel® Resource Director Technology (Intel® RDT) Monitoring capability if 1. Bit 13: Deprecates FPU CS and FPU DS values if 1. Bit 14: MPX. Supports Intel® Memory Protection Extensions if 1. Bit 15: RDT-A. Supports Intel® Resource Director Technology (Intel® RDT) Allocation capability if 1. Bit 16: AVX512F. Bit 17: AVX512DQ. Bit 18: RDSEED. Bit 19: ADX. Bit 20: SMAP. Supports Supervisor-Mode Access Prevention (and the CLAC/STAC instructions) if 1. Bit 21: AVX512_IFMA. Bit 22: Reserved. Bit 23: CLFLUSHOPT. Bit 24: CLWB. Bit 25: Intel Processor Trace. Bit 26: AVX512PF. (Intel® Xeon Phi™ only.) Bit 27: AVX512ER. (Intel® Xeon Phi™ only.) Bit 28: AVX512CD. Bit 29: SHA. supports Intel® Secure Hash Algorithm Extensions (Intel® SHA Extensions) if 1. Bit 30: AVX512BW. Bit 31: AVX512VL.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 00: PREFETCHWT1. (Intel® Xeon Phi™ only.) Bit 01: AVX512_VBMI. Bit 02: UMIP. Supports user-mode instruction prevention if 1. Bit 03: PKU. Supports protection keys for user-mode pages if 1. Bit 04: OSPKE. If 1, OS has set CR4.PKE to enable protection keys (and the RDPKRU/WRPKRU instructions). Bit 05: WAITPKG. Bit 06: AVX512_VBMI2. Bit 07: CET_SS. Supports CET shadow stack features if 1. 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_PL0_SSP. Bit 08: GFNI. Bit 09: VAES. Bit 10: VPCLMULQDQ. Bit 11: AVX512_VNNI. Bit 12: AVX512_BITALG. Bits 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. Bit 14: AVX512_VPOPCNTDQ. Bit 15: Reserved. Bit 16: LA57. Supports 57-bit linear addresses and five-level paging if 1. Bits 21 - 17: The value of MAWAU used by the BNDLDX and BNDSTX instructions in 64-bit mode. Bit 22: RDPID and IA32_TSC_AUX are available if 1. Bit 23: KL. Supports Key Locker if 1. Bit 24: Reserved. Bit 25: CLDEMOTE. Supports cache line demote if 1. Bit 26: Reserved. Bit 27: MOVDIRI. Supports MOVDIRI if 1. Bit 28: MOVDIR64B. Supports MOVDIR64B if 1. Bit 29: Reserved. Bit 30: SGX_LC. Supports SGX Launch Configuration if 1. Bit 31: PKS. Supports protection keys for supervisor-mode pages if 1.
	EDX	Bit 01: Reserved. Bit 02: AVX512_4VNNIW. (Intel® Xeon Phi™ only.) Bit 03: AVX512_4FMAPS. (Intel® Xeon Phi™ only.) Bit 04: Fast Short REP MOV. Bits 07-05: Reserved. Bit 08: AVX512_VP2INTERSECT. Bit 09: Reserved. Bit 10: MD_CLEAR supported. Bits 13-11: Reserved. Bit 14: SERIALIZE. Bit 15: Hybrid. If 1, the processor is identified as a hybrid part. Bits 17-16: Reserved. Bit 18: PCONFIG. Supports PCONFIG if 1. Bit 19: Reserved. Bit 20: CET_IBT. Supports CET indirect branch tracking features if 1. Processors that set this bit define bits 5:2 and bits 63:10 of the IA32_U_CET and IA32_S_CET MSRs. Bits 25 - 21: Reserved. Bit 26: Enumerates support for 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).

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
		Bit 27: Enumerates support for 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). Bit 28: Enumerates support for L1D_FLUSH. Processors that set this bit support the IA32_FLUSH_CMD MSR. They allow software to set IA32_FLUSH_CMD[0] (L1D_FLUSH). Bit 29: Enumerates support for the IA32_ARCH_CAPABILITIES MSR. Bit 30: Enumerates support for 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. Additionally, on hybrid parts (CPUID.07H.0H:EDX[15]=1), software must consult the native model ID and core type from the Hybrid Information Enumeration Leaf. Bit 31: Enumerates support for 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). **NOTE:** * 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.
	Structured Extended Feature Enumeration Sub-leaf (EAX = 07H, ECX = 1)	
07H		**NOTES:** Leaf 07H output depends on the initial value in ECX. If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0.
	EAX	This field reports 0 if the sub-leaf index, *1*, is invalid. Bits 03-00: Reserved. Bit 04: AVX-VNNI. AVX (VEX-encoded) versions of the Vector Neural Network Instructions. Bit 05: AVX512_BF16. Vector Neural Network Instructions supporting BFLOAT16 inputs and conversion instructions from IEEE single precision. Bits 09-06: Reserved. Bit 10: If 1, supports fast zero-length REP MOVSB. Bit 11: If 1, supports fast short REP STOSB. Bit 12: If 1, supports fast short REP CMPSB, REP SCASB. Bits 21-13: Reserved. Bit 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. Bits 31-23: Reserved.
	EBX	This field reports 0 if the sub-leaf index, *1*, is invalid. Bit 00: Enumerates the presence of the IA32_PPIN and IA32_PPIN_CTL MSRs. If 1, these MSRs are supported. Bits 31-01: Reserved.
	ECX	This field reports 0 if the sub-leaf index, *1*, is invalid; otherwise it is reserved.
	EDX	This field reports 0 if the sub-leaf index, *1*, is invalid; otherwise it is reserved.
	Direct Cache Access Information Leaf	
09H	EAX	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H).
	EBX	Reserved.
	ECX	Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Reserved.
	Architectural Performance Monitoring Leaf	
0AH	EAX	Bits 07 - 00: Version ID of architectural performance monitoring. Bits 15 - 08: Number of general-purpose performance monitoring counter per logical processor. Bits 23 - 16: Bit width of general-purpose, performance monitoring counter. Bits 31 - 24: Length of EBX bit vector to enumerate architectural performance monitoring events. Architectural event x is supported if EBX[x]=0 && EAX[31:24]>x.
	EBX	Bit 00: Core cycle event not available if 1 or if EAX[31:24]<1. Bit 01: Instruction retired event not available if 1 or if EAX[31:24]<2. Bit 02: Reference cycles event not available if 1 or if EAX[31:24]<3. Bit 03: Last-level cache reference event not available if 1 or if EAX[31:24]<4. Bit 04: Last-level cache misses event not available if 1 or if EAX[31:24]<5. Bit 05: Branch instruction retired event not available if 1 or if EAX[31:24]<6. Bit 06: Branch mispredict retired event not available if 1 or if EAX[31:24]<7. Bit 07: Top-down slots event not available if 1 or if EAX[31:24]<8. Bits 31 - 08: Reserved = 0.
	ECX	Bits 31 - 00: 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);
	EDX	Bits 04 - 00: Number of contiguous fixed-function performance counters starting from 0 (if Version ID > 1). Bits 12 - 05: Bit width of fixed-function performance counters (if Version ID > 1). Bits 14 - 13: Reserved = 0. Bit 15: AnyThread deprecation. Bits 31 - 16: Reserved = 0.
	Extended Topology Enumeration Leaf	
0BH		**NOTES:** *CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of Leaf 1FH before using leaf 0BH.* Most of Leaf 0BH output depends on the initial value in ECX. The EDX output of leaf 0BH is always valid and does not vary with input value in ECX. Output value in ECX[7:0] always equals input value in ECX[7:0]. Sub-leaf index 0 enumerates SMT level. Each subsequent higher sub-leaf index enumerates a higher-level topological entity in hierarchical order. For sub-leaves that return an invalid level-type of 0 in ECX[15:8]; EAX and EBX will return 0. If an input value n in ECX returns the invalid level-type of 0 in ECX[15:8], other input values with ECX > n also return 0 in ECX[15:8].
	EAX	Bits 04 - 00: Number of bits to shift right on x2APIC ID to get a unique topology ID of the next level type*. All logical processors with the same next level ID share current level. Bits 31 - 05: Reserved.
	EBX	Bits 15 - 00: Number of logical processors at this level type. The number reflects configuration as shipped by Intel**. Bits 31- 16: Reserved.
	ECX	Bits 07 - 00: Level number. Same value in ECX input. Bits 15 - 08: Level type***. Bits 31 - 16: Reserved.
	EDX	Bits 31- 00: x2APIC ID the current logical processor.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
		NOTES: * Software should use this field (EAX[4:0]) to enumerate processor topology of the system. ** Software must not use EBX[15:0] to enumerate processor topology of the system. This value in this field (EBX[15:0]) is only intended for display/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. *** The value of the "level type" field is not related to level numbers in any way, higher "level type" values do not mean higher levels. Level type field has the following encoding: 0: Invalid. 1: SMT. 2: Core. 3-255: Reserved.
Processor Extended State Enumeration Main Leaf (EAX = 0DH, ECX = 0)		
0DH		**NOTES:** Leaf 0DH main leaf (ECX = 0).
	EAX	Bits 31 - 00: Reports the supported bits of the lower 32 bits of XCR0. XCR0[n] can be set to 1 only if EAX[n] is 1. Bit 00: x87 state. Bit 01: SSE state. Bit 02: AVX state. Bits 04 - 03: MPX state. Bits 07 - 05: AVX-512 state. Bit 08: Used for IA32_XSS. Bit 09: PKRU state. Bits 12 - 10: Reserved. Bit 13: Used for IA32_XSS. Bits 15 - 14: Reserved. Bit 16: Used for IA32_XSS. Bits 31 - 17: Reserved.
	EBX	Bits 31 - 00: 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.
	ECX	Bit 31 - 00: 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 processor, i.e., all the valid bit fields in XCR0.
	EDX	Bit 31 - 00: 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. Bits 31 - 00: Reserved.
Processor Extended State Enumeration Sub-leaf (EAX = 0DH, ECX = 1)		
0DH	EAX	Bit 00: XSAVEOPT is available. Bit 01: Supports XSAVEC and the compacted form of XRSTOR if set. Bit 02: Supports XGETBV with ECX = 1 if set. Bit 03: Supports XSAVES/XRSTORS and IA32_XSS if set. Bits 31 - 04: Reserved.
	EBX	Bits 31 - 00: The size in bytes of the XSAVE area containing all states enabled by XCR0 | IA32_XSS.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bits 31 - 00: 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. Bits 07 - 00: Used for XCR0. Bit 08: PT state. Bit 09: Used for XCR0. Bit 10: Reserved. Bit 11: CET user state. Bit 12: CET supervisor state. Bit 13: HDC state. Bit 14: Reserved. Bit 15: LBR state (architectural). Bit 16: HWP state. Bits 31 - 17: Reserved.
	EDX	Bits 31 - 00: 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. Bits 31 - 00: Reserved.
Processor Extended State Enumeration Sub-leaves (EAX = 0DH, ECX = n, n > 1)		
0DH		**NOTES:** Leaf 0DH output depends on the initial value in ECX. Each sub-leaf index (starting at position 2) is supported if it corresponds to a supported bit in either the XCR0 register or the IA32_XSS MSR. * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf n (0 ≤ n ≤ 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 ≤ n ≤ 63) is invalid if sub-leaf 0 returns 0 in EDX[n-32] and sub-leaf 1 returns 0 in EDX[n-32].
	EAX	Bits 31 - 0: 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*.
	EBX	Bits 31 - 0: 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*.
	ECX	Bit 00 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. Bit 01 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). Bits 31 - 02 are reserved. This field reports 0 if the sub-leaf index, n, is invalid*.
	EDX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.
Intel Resource Director Technology (Intel RDT) Monitoring Enumeration Sub-leaf (EAX = 0FH, ECX = 0)		
0FH		**NOTES:** Leaf 0FH output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX.
	EAX	Reserved.
	EBX	Bits 31 - 00: Maximum range (zero-based) of RMID within this physical processor of all types.
	ECX	Reserved.
	EDX	Bit 00: Reserved. Bit 01: Supports L3 Cache Intel RDT Monitoring if 1. Bits 31 - 02: Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	L3 Cache Intel RDT Monitoring Capability Enumeration Sub-leaf (EAX = 0FH, ECX = 1)
0FH	**NOTES:** Leaf 0FH output depends on the initial value in ECX. EAX: Reserved. EBX: Bits 31 - 00: Conversion factor from reported IA32_QM_CTR value to occupancy metric (bytes) and Memory Bandwidth Monitoring (MBM) metrics. ECX: Maximum range (zero-based) of RMID of this resource type. EDX: Bit 00: Supports L3 occupancy monitoring if 1. Bit 01: Supports L3 Total Bandwidth monitoring if 1. Bit 02: Supports L3 Local Bandwidth monitoring if 1. Bits 31 - 03: Reserved.
	Intel Resource Director Technology (Intel RDT) Allocation Enumeration Sub-leaf (EAX = 10H, ECX = 0)
10H	**NOTES:** Leaf 10H output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource identification (ResID) starting at bit position 1 of EBX. EAX: Reserved. EBX: Bit 00: Reserved. Bit 01: Supports L3 Cache Allocation Technology if 1. Bit 02: Supports L2 Cache Allocation Technology if 1. Bit 03: Supports Memory Bandwidth Allocation if 1. Bits 31 - 04: Reserved. ECX: Reserved. EDX: Reserved.
	L3 Cache Allocation Technology Enumeration Sub-leaf (EAX = 10H, ECX = ResID =1)
10H	**NOTES:** Leaf 10H output depends on the initial value in ECX. EAX: Bits 04 - 00: Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result. Bits 31 - 05: Reserved. EBX: Bits 31 - 00: Bit-granular map of isolation/contention of allocation units. ECX: Bits 01- 00: Reserved. Bit 02: Code and Data Prioritization Technology supported if 1. Bits 31 - 03: Reserved. EDX: Bits 15 - 00: Highest COS number supported for this ResID. Bits 31 - 16: Reserved.
	L2 Cache Allocation Technology Enumeration Sub-leaf (EAX = 10H, ECX = ResID =2)
10H	**NOTES:** Leaf 10H output depends on the initial value in ECX. EAX: Bits 04 - 00: Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result. Bits 31 - 05: Reserved. EBX: Bits 31 - 00: Bit-granular map of isolation/contention of allocation units. ECX: Bits 31 - 00: Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 15 - 00: Highest COS number supported for this ResID. Bits 31 - 16: Reserved.
	Memory Bandwidth Allocation Enumeration Sub-leaf (EAX = 10H, ECX = ResID =3)	
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX.
	EAX	Bits 11 - 00: Reports the maximum MBA throttling value supported for the corresponding ResID. Add one to the return value to get the result. Bits 31 - 12: Reserved.
	EBX	Bits 31 - 00: Reserved.
	ECX	Bits 01 - 00: Reserved. Bit 02: Reports whether the response of the delay values is linear. Bits 31 - 03: Reserved.
	EDX	Bits 15 - 00: Highest COS number supported for this ResID. Bits 31 - 16: Reserved.
	Intel SGX Capability Enumeration Leaf, sub-leaf 0 (EAX = 12H, ECX = 0)	
12H		**NOTES:** Leaf 12H sub-leaf 0 (ECX = 0) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.
	EAX	Bit 00: SGX1. If 1, Indicates Intel SGX supports the collection of SGX1 leaf functions. Bit 01: SGX2. If 1, Indicates Intel SGX supports the collection of SGX2 leaf functions. Bits 04 - 02: Reserved. Bit 05: If 1, indicates Intel SGX supports ENCLV instruction leaves EINCVIRTCHILD, EDECVIRTCHILD, and ESETCONTEXT. Bit 06: If 1, indicates Intel SGX supports ENCLS instruction leaves ETRACKC, ERDINFO, ELDBC, and ELDUC. Bits 31 - 07: Reserved.
	EBX	Bits 31 - 00: MISCSELECT. Bit vector of supported extended SGX features.
	ECX	Bits 31 - 00: Reserved.
	EDX	Bits 07 - 00: MaxEnclaveSize_Not64. The maximum supported enclave size in non-64-bit mode is 2^(EDX[7:0]). Bits 15 - 08: MaxEnclaveSize_64. The maximum supported enclave size in 64-bit mode is 2^(EDX[15:8]). Bits 31 - 16: Reserved.
	Intel SGX Attributes Enumeration Leaf, sub-leaf 1 (EAX = 12H, ECX = 1)	
12H		**NOTES:** Leaf 12H sub-leaf 1 (ECX = 1) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.
	EAX	Bit 31 - 00: Reports the valid bits of SECS.ATTRIBUTES[31:0] that software can set with ECREATE.
	EBX	Bit 31 - 00: Reports the valid bits of SECS.ATTRIBUTES[63:32] that software can set with ECREATE.
	ECX	Bit 31 - 00: Reports the valid bits of SECS.ATTRIBUTES[95:64] that software can set with ECREATE.
	EDX	Bit 31 - 00: Reports the valid bits of SECS.ATTRIBUTES[127:96] that software can set with ECREATE.
	Intel SGX EPC Enumeration Leaf, sub-leaves (EAX = 12H, ECX = 2 or higher)	
12H		**NOTES:** Leaf 12H sub-leaf 2 or higher (ECX >= 2) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1. For sub-leaves (ECX = 2 or higher), definition of EDX,ECX,EBX,EAX[31:4] depends on the sub-leaf type listed below.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	EAX Bit 03 - 00: Sub-leaf Type 0000b: Indicates this sub-leaf is invalid. 0001b: This sub-leaf enumerates an EPC section. EBX:EAX and EDX:ECX provide information on the Enclave Page Cache (EPC) section. All other type encodings are reserved. Type 0000b. This sub-leaf is invalid. EDX:ECX:EBX:EAX return 0. Type 0001b. This sub-leaf enumerates an EPC sections with EDX:ECX, EBX:EAX defined as follows. EAX[11:04]: Reserved (enumerate 0). EAX[31:12]: Bits 31:12 of the physical address of the base of the EPC section. EBX[19:00]: Bits 51:32 of the physical address of the base of the EPC section. EBX[31:20]: Reserved. ECX[03:00]: EPC section property encoding defined as follows: If ECX[3:0] = 0000b, then all bits of the EDX:ECX pair are enumerated as 0. If ECX[3:0] = 0001b, then this section has confidentiality and integrity protection. If ECX[3:0] = 0010b, then this section has confidentiality protection only. All other encodings are reserved. ECX[11:04]: Reserved (enumerate 0). ECX[31:12]: Bits 31:12 of the size of the corresponding EPC section within the Processor Reserved Memory. EDX[19:00]: Bits 51:32 of the size of the corresponding EPC section within the Processor Reserved Memory. EDX[31:20]: Reserved.
	Intel Processor Trace Enumeration Main Leaf (EAX = 14H, ECX = 0)
14H	**NOTES:** Leaf 14H main leaf (ECX = 0). EAX Bits 31 - 00: Reports the maximum sub-leaf supported in leaf 14H. EBX Bit 00: If 1, indicates that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed. Bit 01: If 1, indicates support of Configurable PSB and Cycle-Accurate Mode. Bit 02: If 1, indicates support of IP Filtering, TraceStop filtering, and preservation of Intel PT MSRs across warm reset. Bit 03: If 1, indicates support of MTC timing packet and suppression of COFI-based packets. Bit 04: If 1, indicates support of PTWRITE. Writes can set IA32_RTIT_CTL[12] (PTWEn) and IA32_RTIT_CTL[5] (FUPonPTW), and PTWRITE can generate packets. Bit 05: If 1, indicates support of Power Event Trace. Writes can set IA32_RTIT_CTL[4] (PwrEvtEn), enabling Power Event Trace packet generation. Bit 06: If 1, indicates support for PSB and PMI preservation. 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. Bit 07: If 1, writes can set IA32_RTIT_CTL[31] (EventEn), enabling Event Trace packet generation. Bit 08: If 1, writes can set IA32_RTIT_CTL[55] (DisTNT), disabling TNT packet generation. Bit 31 - 09: Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 00: If 1, Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme; IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed. Bit 01: If 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. Bit 02: If 1, indicates support of Single-Range Output scheme. Bit 03: If 1, indicates support of output to Trace Transport subsystem. Bit 30 - 04: Reserved. Bit 31: If 1, generated packets which contain IP payloads have LIP values, which include the CS base component.
	EDX	Bits 31 - 00: Reserved.
	Intel Processor Trace Enumeration Sub-leaf (EAX = 14H, ECX = 1)	
14H	EAX	Bits 02 - 00: Number of configurable Address Ranges for filtering. Bits 15 - 03: Reserved. Bits 31 - 16: Bitmap of supported MTC period encodings.
	EBX	Bits 15 - 00: Bitmap of supported Cycle Threshold value encodings. Bit 31 - 16: Bitmap of supported Configurable PSB frequency encodings.
	ECX	Bits 31 - 00: Reserved.
	EDX	Bits 31 - 00: Reserved.
	Time Stamp Counter and Nominal Core Crystal Clock Information Leaf	
15H		**NOTES:** If EBX[31:0] is 0, the TSC/"core crystal clock" ratio is not enumerated. EBX[31:0]/EAX[31:0] indicates the ratio of the TSC frequency and the core crystal clock frequency. If ECX is 0, the nominal core crystal clock frequency is not enumerated. "TSC frequency" = "core crystal clock frequency" * EBX/EAX. The core crystal clock may differ from the reference clock, bus clock, or core clock frequencies.
	EAX	Bits 31 - 00: An unsigned integer which is the denominator of the TSC/"core crystal clock" ratio.
	EBX	Bits 31 - 00: An unsigned integer which is the numerator of the TSC/"core crystal clock" ratio.
	ECX	Bits 31 - 00: An unsigned integer which is the nominal frequency of the core crystal clock in Hz.
	EDX	Bits 31 - 00: Reserved = 0.
	Processor Frequency Information Leaf	
16H	EAX	Bits 15 - 00: Processor Base Frequency (in MHz). Bits 31 - 16: Reserved =0.
	EBX	Bits 15 - 00: Maximum Frequency (in MHz). Bits 31 - 16: Reserved = 0.
	ECX	Bits 15 - 00: Bus (Reference) Frequency (in MHz). Bits 31 - 16: Reserved = 0.
	EDX	Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	NOTES: * 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.
System-On-Chip Vendor Attribute Enumeration Main Leaf (EAX = 17H, ECX = 0)	
17H	**NOTES:** Leaf 17H main leaf (ECX = 0). Leaf 17H output depends on the initial value in ECX. Leaf 17H sub-leaves 1 through 3 reports SOC Vendor Brand String. Leaf 17H is valid if MaxSOCID_Index >= 3. Leaf 17H sub-leaves 4 and above are reserved. EAX: Bits 31 - 00: MaxSOCID_Index. Reports the maximum input value of supported sub-leaf in leaf 17H. EBX: Bits 15 - 00: SOC Vendor ID. Bit 16: IsVendorScheme. 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. Bits 31 - 17: Reserved = 0. ECX: Bits 31 - 00: Project ID. A unique number an SOC vendor assigns to its SOC projects. EDX: Bits 31 - 00: Stepping ID. A unique number within an SOC project that an SOC vendor assigns.
System-On-Chip Vendor Attribute Enumeration Sub-leaf (EAX = 17H, ECX = 1..3)	
17H	EAX: Bit 31 - 00: SOC Vendor Brand String. UTF-8 encoded string. EBX: Bit 31 - 00: SOC Vendor Brand String. UTF-8 encoded string. ECX: Bit 31 - 00: SOC Vendor Brand String. UTF-8 encoded string. EDX: Bit 31 - 00: SOC Vendor Brand String. UTF-8 encoded string. **NOTES:** Leaf 17H output depends on the initial value in ECX. 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 of EAX:EBX:ECX:EDX and from the sub-leaf 1 fragment towards sub-leaf 3.
System-On-Chip Vendor Attribute Enumeration Sub-leaves (EAX = 17H, ECX > MaxSOCID_Index)	
17H	**NOTES:** Leaf 17H output depends on the initial value in ECX. EAX: Bits 31 - 00: Reserved = 0. EBX: Bits 31 - 00: Reserved = 0. ECX: Bits 31 - 00: Reserved = 0. EDX: Bits 31 - 00: Reserved = 0.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
Deterministic Address Translation Parameters Main Leaf (EAX = 18H, ECX = 0)	
18H	**NOTES:** Each sub-leaf enumerates a different address translation structure. 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. * 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) . Please see the *Intel® 64 and IA-32 Architectures Optimization Reference Manual* for details of a particular product. ** Add one to the return value to get the result. EAX Bits 31 - 00: Reports the maximum input value of supported sub-leaf in leaf 18H. EBX Bit 00: 4K page size entries supported by this structure. Bit 01: 2MB page size entries supported by this structure. Bit 02: 4MB page size entries supported by this structure. Bit 03: 1 GB page size entries supported by this structure. Bits 07 - 04: Reserved. Bits 10 - 08: Partitioning (0: Soft partitioning between the logical processors sharing this structure). Bits 15 - 11: Reserved. Bits 31 - 16: W = Ways of associativity. ECX Bits 31 - 00: S = Number of Sets. EDX Bits 04 - 00: Translation cache type field. 00000b: Null (indicates this sub-leaf is not valid). 00001b: Data TLB. 00010b: Instruction TLB. 00011b: Unified TLB*. 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. Bits 07 - 05: Translation cache level (starts at 1). Bit 08: Fully associative structure. Bits 13 - 09: Reserved. Bits 25- 14: Maximum number of addressable IDs for logical processors sharing this translation cache** Bits 31 - 26: Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	Deterministic Address Translation Parameters Sub-leaf (EAX = 18H, ECX ≥ 1)	
18H	**NOTES:** Each sub-leaf enumerates a different address translation structure. 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. * 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. ** Add one to the return value to get the result.	
	EAX	Bits 31 - 00: Reserved.
	EBX	Bit 00: 4K page size entries supported by this structure. Bit 01: 2MB page size entries supported by this structure. Bit 02: 4MB page size entries supported by this structure. Bit 03: 1 GB page size entries supported by this structure. Bits 07 - 04: Reserved. Bits 10 - 08: Partitioning (0: Soft partitioning between the logical processors sharing this structure). Bits 15 - 11: Reserved. Bits 31 - 16: W = Ways of associativity.
	ECX	Bits 31 - 00: S = Number of Sets.
	EDX	Bits 04 - 00: Translation cache type field. 0000b: Null (indicates this sub-leaf is not valid). 0001b: Data TLB. 0010b: Instruction TLB. 0011b: Unified TLB*. All other encodings are reserved. Bits 07 - 05: Translation cache level (starts at 1). Bit 08: Fully associative structure. Bits 13 - 09: Reserved. Bits 25- 14: Maximum number of addressable IDs for logical processors sharing this translation cache** Bits 31 - 26: Reserved.
	Key Locker Leaf (EAX = 19H)	
19H	EAX	Bit 00: Key Locker restriction of CPL0-only supported. Bit 01: Key Locker restriction of no-encrypt supported. Bit 02: Key Locker restriction of no-decrypt supported. Bits 31-03: Reserved.
	EBX	Bit 00: AESKLE. If 1, the AES Key Locker instructions are fully enabled. Bit 01: Reserved. Bit 02: If 1, the AES wide Key Locker instructions are supported. Bit 03: Reserved. Bit 04: If 1, the platform 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. Bits 31-05: Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 00: If 1, the NoBackup parameter to LOADIWKEY is supported. Bit 01: If 1, KeySource encoding of 1 (randomization of the internal wrapping key) is supported. Bits 31- 02: Reserved.
	EDX	Reserved.
Hybrid Information Enumeration Leaf (EAX = 1AH, ECX = 0)		
1AH	EAX	Enumerates the native model ID and core type. Bits 31-24: Core type 10H: Reserved 20H: Intel Atom® 30H: Reserved 40H: Intel® Core™ Bits 23-0: Native model ID of the core. The core-type and native mode 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 leaf 01H, and does not identify the SOC.
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.
PCONFIG Information Sub-leaf (EAX = 1BH, ECX ≥ 0)		
1BH		For details on this sub-leaf, see "INPUT EAX = 1BH: Returns PCONFIG Information" on page 3-247. **NOTE:** Leaf 1BH is supported if CPUID.(EAX=07H, ECX=0H):EDX[18] = 1.
Last Branch Records Information Leaf (EAX = 1CH, ECX = 0)		
1CH		**NOTES:** This leaf pertains to the architectural feature. For leaf 01CH, CPUID will ignore the ECX value.
	EAX	Bits 07 - 00: Supported LBR Depth Values. For each bit n set in this field, the IA32_LBR_DEPTH.DEPTH value 8*(n+1) is supported. Bits 29 - 08: Reserved. Bit 30: Deep C-state Reset. If set, indicates that LBRs may be cleared on an MWAIT that requests a C-state numerically greater than C1. Bit 31: IP Values Contain LIP. If set, LBR IP values contain LIP. If clear, IP values contain Effective IP.
	EBX	Bit 00: CPL Filtering Supported. If set, the processor supports setting IA32_LBR_CTL[2:1] to non-zero value. Bit 01: Branch Filtering Supported. If set, the processor supports setting IA32_LBR_CTL[22:16] to non-zero value. Bit 02: Call-stack Mode Supported. If set, the processor supports setting IA32_LBR_CTL[3] to 1. Bits 31 - 03: Reserved.
	ECX	Bit 00: Mispredict Bit Supported. IA32_LBR_x_INFO[63] holds indication of branch misprediction (MISPRED). Bit 01: Timed LBRs Supported. 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). Bit 02: Branch Type Field Supported. IA32_LBR_INFO_x[59:56] holds indication of the recorded operation's branch type (BR_TYPE). Bits 31 - 03: Reserved.
	EDX	Bits 31 - 00: Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	V2 Extended Topology Enumeration Leaf	
1FH		**NOTES:** *CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of Leaf 1FH and using this if available.* Most of Leaf 1FH output depends on the initial value in ECX. The EDX output of leaf 1FH is always valid and does not vary with input value in ECX. Output value in ECX[7:0] always equals input value in ECX[7:0]. Sub-leaf index 0 enumerates SMT level. Each subsequent higher sub-leaf index enumerates a higher-level topological entity in hierarchical order. For sub-leaves that return an invalid level-type of 0 in ECX[15:8]; EAX and EBX will return 0. If an input value n in ECX returns the invalid level-type of 0 in ECX[15:8], other input values with ECX > n also return 0 in ECX[15:8].
	EAX	Bits 04 - 00: Number of bits to shift right on x2APIC ID to get a unique topology ID of the next level type*. All logical processors with the same next level ID share current level. Bits 31 - 05: Reserved.
	EBX	Bits 15 - 00: Number of logical processors at this level type. The number reflects configuration as shipped by Intel**. Bits 31- 16: Reserved.
	ECX	Bits 07 - 00: Level number. Same value in ECX input. Bits 15 - 08: Level type***. Bits 31 - 16: Reserved.
	EDX	Bits 31- 00: x2APIC ID the current logical processor.
		NOTES: * Software should use this field (EAX[4:0]) to enumerate processor topology of the system. ** Software must not use EBX[15:0] to enumerate processor topology of the system. This value in this field (EBX[15:0]) is only intended for display/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. *** The value of the "level type" field is not related to level numbers in any way, higher "level type" values do not mean higher levels. Level type field has the following encoding: 0: Invalid. 1: SMT. 2: Core. 3: Module. 4: Tile. 5: Die. 6-255: Reserved.
	Processor History Reset Sub-leaf (EAX = 20H, ECX = 0)	
20H	EAX	Reports the maximum number of sub-leaves that are supported in leaf 20H.
	EBX	Indicates which bits may be set in the IA32_HRESET_ENABLE MSR to enable reset of different components of hardware-maintained history. Bit 00: 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. Bits 31-01: Reserved = 0.
	ECX	Reserved.
	EDX	Reserved.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	Unimplemented CPUID Leaf Functions	
21H		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is 21H. If the value returned by CPUID.0:EAX (the maximum input value for basic CPUID information) is at least 21H, 0 is returned in the registers EAX, EBX, ECX, and EDX. Otherwise, the data for the highest basic information leaf is returned.
40000000H - 4FFFFFFFH		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is in the range 40000000H to 4FFFFFFFH.
	Extended Function CPUID Information	
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information.
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.
80000001H	EAX	Extended Processor Signature and Feature Bits.
	EBX	Reserved.
	ECX	Bit 00: LAHF/SAHF available in 64-bit mode.* Bits 04 - 01: Reserved. Bit 05: LZCNT. Bits 07 - 06: Reserved. Bit 08: PREFETCHW. Bits 31 - 09: Reserved.
	EDX	Bits 10 - 00: Reserved. Bit 11: SYSCALL/SYSRET.** Bits 19 - 12: Reserved = 0. Bit 20: Execute Disable Bit available. Bits 25 - 21: Reserved = 0. Bit 26: 1-GByte pages are available if 1. Bit 27: RDTSCP and IA32_TSC_AUX are available if 1. Bit 28: Reserved = 0. Bit 29: Intel® 64 Architecture available if 1. Bits 31 - 30: Reserved = 0. **NOTES:** * LAHF and SAHF are always available in other modes, regardless of the enumeration of this feature flag. ** Intel processors support SYSCALL and SYSRET only in 64-bit mode. This feature flag is always enumerated as 0 outside 64-bit mode.
80000002H	EAX EBX ECX EDX	Processor Brand String. Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued.
80000003H	EAX EBX ECX EDX	Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued.

Table 3-8. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
80000004H	EAX Processor Brand String Continued. EBX Processor Brand String Continued. ECX Processor Brand String Continued. EDX Processor Brand String Continued.
80000005H	EAX Reserved = 0. EBX Reserved = 0. ECX Reserved = 0. EDX Reserved = 0.
80000006H	EAX Reserved = 0. EBX Reserved = 0. ECX Bits 07 - 00: Cache Line size in bytes. Bits 11 - 08: Reserved. Bits 15 - 12: L2 Associativity field *. Bits 31 - 16: Cache size in 1K units. EDX Reserved = 0. **NOTES:** * L2 associativity field encodings: 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 04H, sub-leaf 2**; 0FH - Fully associative ** CPUID leaf 04H provides details of deterministic cache parameters, including the L2 cache in sub-leaf 2
80000007H	EAX Reserved = 0. EBX Reserved = 0. ECX Reserved = 0. EDX Bits 07 - 00: Reserved = 0. Bit 08: Invariant TSC available if 1. Bits 31 - 09: Reserved = 0.
80000008H	EAX Linear/Physical Address size. Bits 07 - 00: #Physical Address Bits*. Bits 15 - 08: #Linear Address Bits. Bits 31 - 16: Reserved = 0. EBX Bits 08-00: Reserved = 0. Bit 09: WBNOINVD is available if 1. Bits 31-10: Reserved = 0. ECX Reserved = 0. EDX Reserved = 0. **NOTES:** * If CPUID.80000008H:EAX[7:0] is supported, the maximum physical address number supported should come from this field.

INPUT EAX = 0: Returns CPUID's Highest Value for Basic Processor Information and the Vendor Identification String

When CPUID executes with EAX set to 0, the processor returns the highest value the CPUID recognizes for returning basic processor information. The value is returned in the EAX register and is processor specific.

A vendor identification string is also returned in EBX, EDX, and ECX. For Intel processors, the string is "Genuin-eIntel" and is expressed:

```
EBX := 756e6547h (* "Genu", with G in the low eight bits of BL *)
EDX := 49656e69h (* "inel", with i in the low eight bits of DL *)
ECX := 6c65746eh (* "ntel", with n in the low eight bits of CL *)
```

INPUT EAX = 80000000H: Returns CPUID's Highest Value for Extended Processor Information

When CPUID executes with EAX set to 80000000H, the processor returns the highest value the processor recognizes for returning extended processor information. The value is returned in the EAX register and is processor specific.

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 CPUID executes. The signature is returned in the upper DWORD. For details, see Chapter 9 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

INPUT EAX = 01H: Returns Model, Family, Stepping Information

When CPUID executes with EAX set to 01H, version information is returned in EAX (see Figure 3-6). 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 3-9 for available processor type values. Stepping IDs are provided as needed.

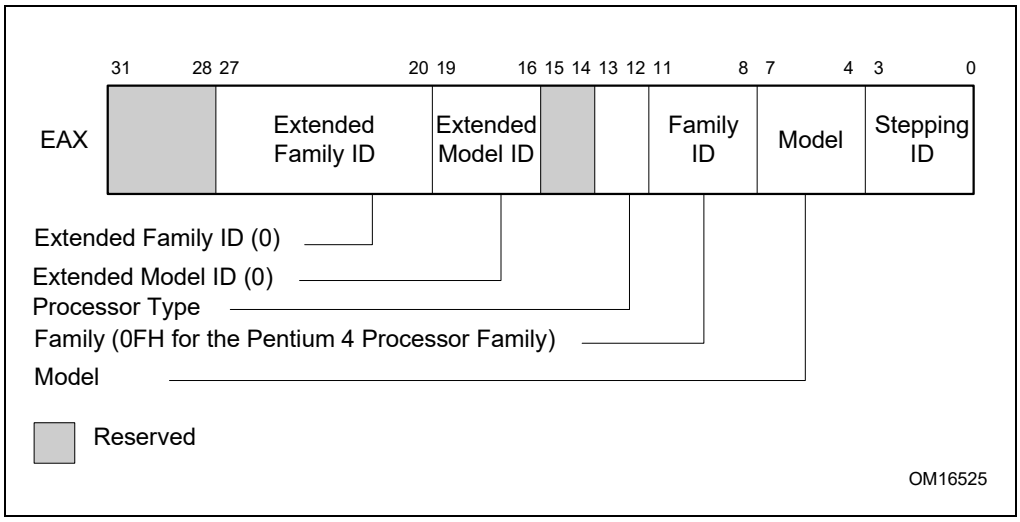


Figure 3-6. Version Information Returned by CPUID in EAX

Table 3-9. 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 Chapter 20 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, 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;
    (* Right justify and zero-extend 4-bit field. *)
FI;
(* Show DisplayFamily as HEX field. *)
```

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;
    (* Right justify and zero-extend 4-bit field; display Model_ID as HEX field.*)
    ELSE DisplayModel = Model_ID;
FI;
(* Show DisplayModel as HEX field. *)
```

INPUT EAX = 01H: Returns Additional Information in EBX

When CPUID executes with EAX set to 01H, additional information is returned to the EBX register:

- Brand index (low byte of EBX) — 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 later in this section.
- CLFLUSH instruction cache line size (second byte of EBX) — 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.
- Local APIC ID (high byte of EBX) — 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.

INPUT EAX = 01H: Returns Feature Information in ECX and EDX

When CPUID executes with EAX set to 01H, feature information is returned in ECX and EDX.

- Figure 3-7 and Table 3-10 show encodings for ECX.
- Figure 3-8 and Table 3-11 show encodings for EDX.

For all feature flags, a 1 indicates that the feature is supported. Use Intel to properly interpret feature flags.

NOTE

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.

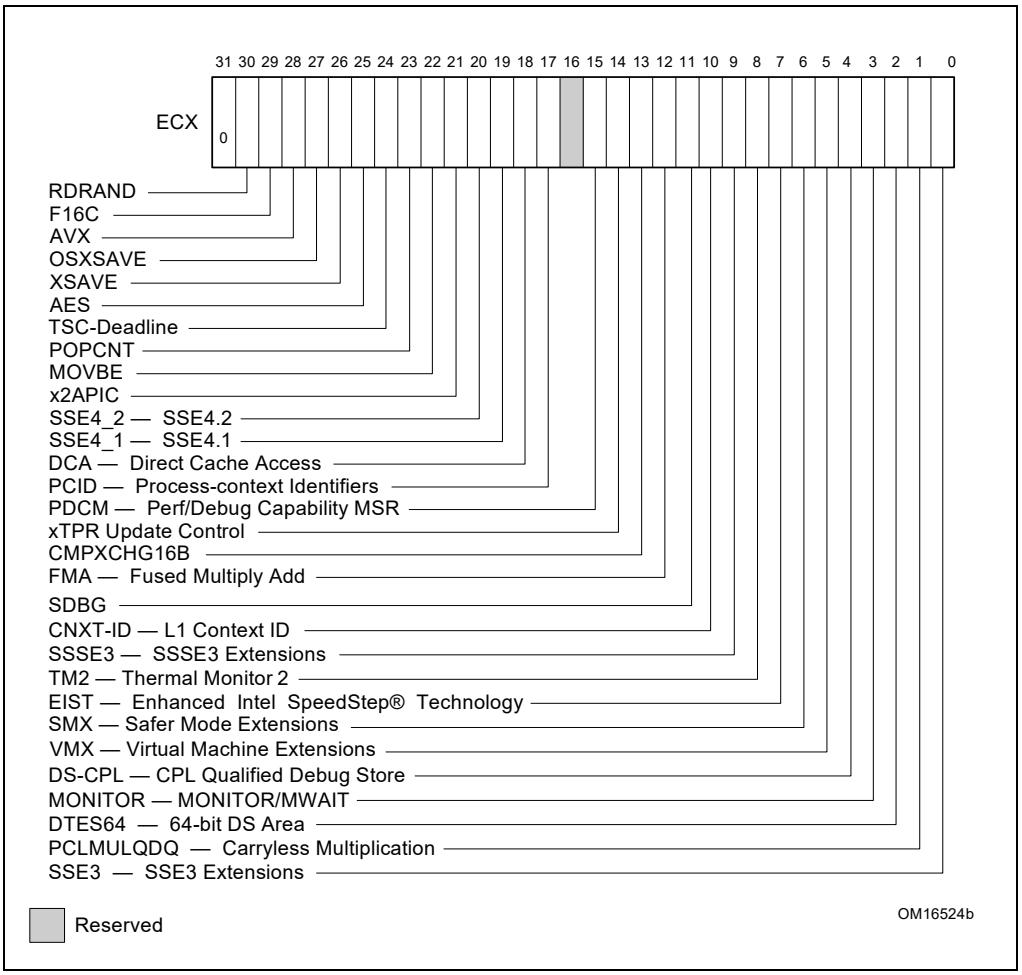


Figure 3-7. Feature Information Returned in the ECX Register

Table 3-10. Feature Information Returned in the ECX Register

Bit #	Mnemonic	Description
0	SSE3	**Streaming SIMD Extensions 3 (SSE3).** A value of 1 indicates the processor supports this technology.
1	PCLMULQDQ	**PCLMULQDQ.** A value of 1 indicates the processor supports the PCLMULQDQ instruction.
2	DTES64	**64-bit DS Area.** A value of 1 indicates the processor supports DS area using 64-bit layout.
3	MONITOR	**MONITOR/MWAIT.** A value of 1 indicates the processor supports this feature.
4	DS-CPL	**CPL Qualified Debug Store.** A value of 1 indicates the processor supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.
5	VMX	**Virtual Machine Extensions.** A value of 1 indicates that the processor supports this technology.
6	SMX	**Safer Mode Extensions.** A value of 1 indicates that the processor supports this technology. See Chapter 6, “Safer Mode Extensions Reference”.
7	EIST	**Enhanced Intel SpeedStep® technology.** A value of 1 indicates that the processor supports this technology.
8	TM2	**Thermal Monitor 2.** A value of 1 indicates whether the processor supports this technology.
9	SSSE3	A value of 1 indicates the presence of the Supplemental Streaming SIMD Extensions 3 (SSSE3). A value of 0 indicates the instruction extensions are not present in the processor.

Table 3-10. Feature Information Returned in the ECX Register (Contd.)

Bit #	Mnemonic	Description
10	CNXT-ID	**L1 Context ID.** A value of 1 indicates the L1 data cache mode can be set to either adaptive mode or shared mode. A value of 0 indicates this feature is not supported. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.
11	SDBG	A value of 1 indicates the processor supports IA32_DEBUG_INTERFACE MSR for silicon debug.
12	FMA	A value of 1 indicates the processor supports FMA extensions using YMM state.
13	CMPXCHG16B	**CMPXCHG16B Available.** A value of 1 indicates that the feature is available. See the "CMPXCHG8B/CMPXCHG16B—Compare and Exchange Bytes" section in this chapter for a description.
14	xTPR Update Control	**xTPR Update Control.** A value of 1 indicates that the processor supports changing IA32_MISC_ENABLE[bit 23].
15	PDCM	**Perfmon and Debug Capability:** A value of 1 indicates the processor supports the performance and debug feature indication MSR IA32_PERF_CAPABILITIES.
16	Reserved	Reserved
17	PCID	**Process-context identifiers.** A value of 1 indicates that the processor supports PCIDs and that software may set CR4.PCIDE to 1.
18	DCA	A value of 1 indicates the processor supports the ability to prefetch data from a memory mapped device.
19	SSE4_1	A value of 1 indicates that the processor supports SSE4.1.
20	SSE4_2	A value of 1 indicates that the processor supports SSE4.2.
21	x2APIC	A value of 1 indicates that the processor supports x2APIC feature.
22	MOVBE	A value of 1 indicates that the processor supports MOVBE instruction.
23	POPCNT	A value of 1 indicates that the processor supports the POPCNT instruction.
24	TSC-Deadline	A value of 1 indicates that the processor's local APIC timer supports one-shot operation using a TSC deadline value.
25	AESNI	A value of 1 indicates that the processor supports the AESNI instruction extensions.
26	XSAVE	A value of 1 indicates that the processor supports the XSAVE/XRSTOR processor extended states feature, the XSETBV/XGETBV instructions, and XCR0.
27	OSXSAVE	A value of 1 indicates that the OS has set CR4.OSXSAVE[bit 18] to enable XSETBV/XGETBV instructions to access XCR0 and to support processor extended state management using XSAVE/XRSTOR.
28	AVX	A value of 1 indicates the processor supports the AVX instruction extensions.
29	F16C	A value of 1 indicates that processor supports 16-bit floating-point conversion instructions.
30	RDRAND	A value of 1 indicates that processor supports RDRAND instruction.
31	Not Used	Always returns 0.

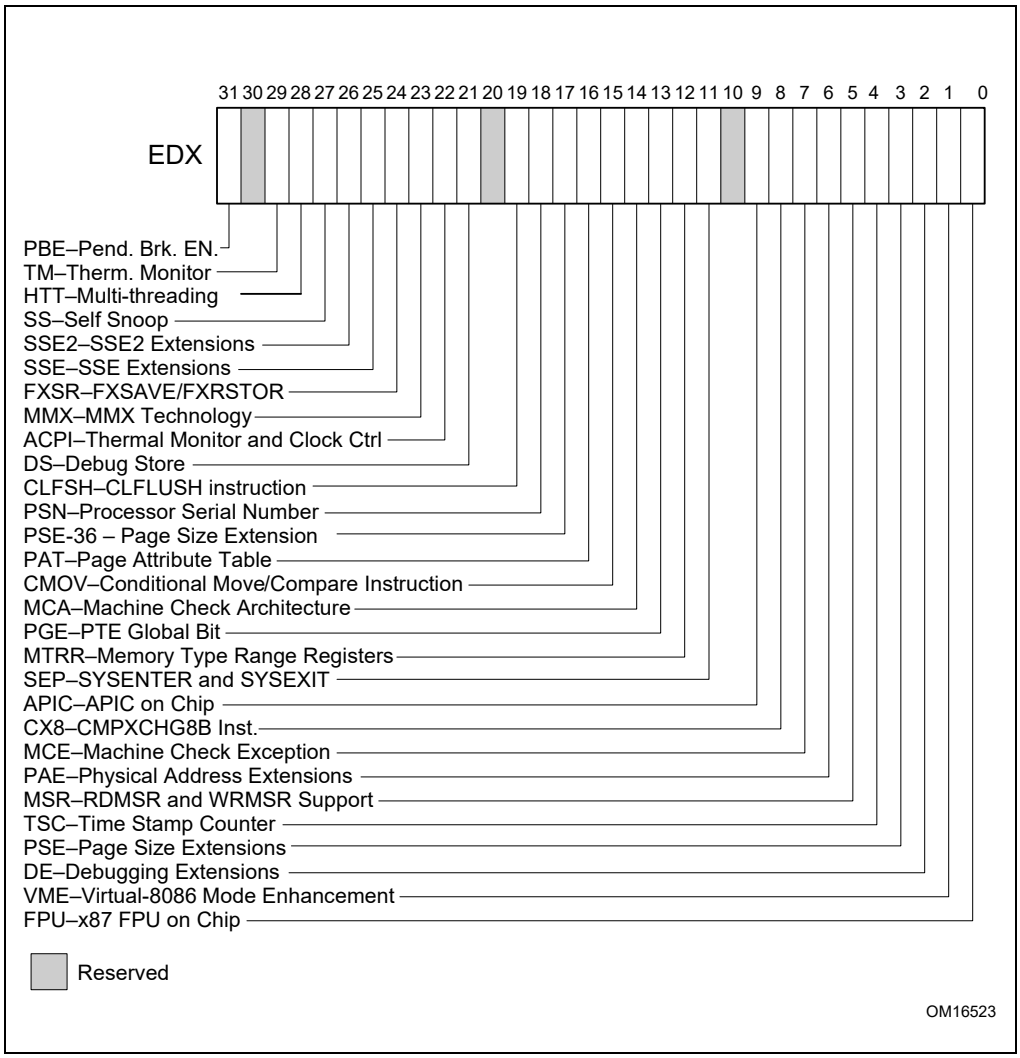


Figure 3-8. Feature Information Returned in the EDX Register

Table 3-11. More on Feature Information Returned in the EDX Register

Bit #	Mnemonic	Description
0	FPU	**Floating Point Unit On-Chip.** The processor contains an x87 FPU.
1	VME	**Virtual 8086 Mode Enhancements.** 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.
2	DE	**Debugging Extensions.** Support for I/O breakpoints, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.
3	PSE	**Page Size Extension.** Large pages of size 4 MByte are supported, 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.
4	TSC	**Time Stamp Counter.** The RDTSC instruction is supported, including CR4.TSD for controlling privilege.
5	MSR	**Model Specific Registers RDMSR and WRMSR Instructions.** The RDMSR and WRMSR instructions are supported. Some of the MSRs are implementation dependent.
6	PAE	**Physical Address Extension.** Physical addresses greater than 32 bits are supported: extended page table entry formats, an extra level in the page translation tables is defined, 2-MByte pages are supported instead of 4 Mbyte pages if PAE bit is 1.
7	MCE	**Machine Check Exception.** Exception 18 is defined for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific 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.
8	CX8	**CMPXCHG8B Instruction.** The compare-and-exchange 8 bytes (64 bits) instruction is supported (implicitly locked and atomic).
9	APIC	**APIC On-Chip.** The processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FFFE0000H to FFFE0FFFH (by default - some processors permit the APIC to be relocated).
10	Reserved	Reserved
11	SEP	**SYSENTER and SYSEXIT Instructions.** The SYSENTER and SYSEXIT and associated MSRs are supported.
12	MTRR	**Memory Type Range Registers.** MTRRs are supported. 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.
13	PGE	**Page Global Bit.** The global bit is supported 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.
14	MCA	**Machine Check Architecture.** A value of 1 indicates the Machine Check Architecture of reporting machine errors is supported. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.
15	CMOV	**Conditional Move Instructions.** The conditional move instruction CMOV is supported. In addition, if x87 FPU is present as indicated by the CPUID.FPU feature bit, then the FCOMI and FCMOV instructions are supported
16	PAT	**Page Attribute Table.** Page Attribute Table is supported. 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.
17	PSE-36	**36-Bit Page Size Extension.** 4-MByte pages addressing physical memory beyond 4 GBytes are supported 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.
18	PSN	**Processor Serial Number.** The processor supports the 96-bit processor identification number feature and the feature is enabled.
19	CLFSH	**CLFLUSH Instruction.** CLFLUSH Instruction is supported.
20	Reserved	Reserved

Table 3-11. More on Feature Information Returned in the EDX Register (Contd.)

Bit #	Mnemonic	Description
21	DS	**Debug Store.** The processor supports 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 23, "Introduction to Virtual-Machine Extensions," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*).
22	ACPI	**Thermal Monitor and Software Controlled Clock Facilities.** The processor implements internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.
23	MMX	**Intel MMX Technology.** The processor supports the Intel MMX technology.
24	FXSR	**FXSAVE and FXRSTOR Instructions.** The FXSAVE and FXRSTOR instructions are supported for fast save and restore of the floating point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it supports the FXSAVE and FXRSTOR instructions.
25	SSE	**SSE.** The processor supports the SSE extensions.
26	SSE2	**SSE2.** The processor supports the SSE2 extensions.
27	SS	**Self Snoop.** The processor supports the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.
28	HTT	**Max APIC IDs reserved field is Valid.** A value of 0 for HTT indicates there is only a single logical processor in the package and software should assume only a single APIC ID is reserved. A value of 1 for HTT indicates 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.
29	TM	**Thermal Monitor.** The processor implements the thermal monitor automatic thermal control circuitry (TCC).
30	Reserved	Reserved
31	PBE	**Pending Break Enable.** The processor supports 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.

INPUT EAX = 02H: TLB/Cache/Prefetch Information Returned in EAX, EBX, ECX, EDX

When CPUID executes with EAX set to 02H, the processor 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 type is noted in the second column of Table 3-12. Table 3-12 lists the encoding of these descriptors. 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" via CPUID leaf 2.

Table 3-12. Encoding of CPUID Leaf 2 Descriptors

Value	Type	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

Table 3-12. Encoding of CPUID Leaf 2 Descriptors (Contd.)

Value	Type	Description
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-µop, 8-way set associative
71H	Cache	Trace cache: 16 K-µop, 8-way set associative
72H	Cache	Trace cache: 32 K-µop, 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, 64byte 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, 64byte 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

Table 3-12. Encoding of CPUID Leaf 2 Descriptors (Contd.)

Value	Type	Description
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 1GBbyte 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 3-1. 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-μop, 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.

INPUT EAX = 04H: Returns Deterministic Cache Parameters for Each Level

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. The architecturally defined fields reported by deterministic cache parameters are documented in Table 3-8.

This Cache Size in Bytes

= (Ways + 1) * (Partitions + 1) * (Line_Size + 1) * (Sets + 1)

= (EBX[31:22] + 1) * (EBX[21:12] + 1) * (EBX[11:0] + 1) * (ECX + 1)

The CPUID leaf 04H also reports data that can be used to derive the topology of processor cores in a physical package. 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 described in Chapter 8, "Multiple-Processor Management," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

INPUT EAX = 05H: Returns MONITOR and MWAIT Features

When CPUID executes with EAX set to 05H, the processor 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. See Table 3-8.

INPUT EAX = 06H: Returns Thermal and Power Management Features

When CPUID executes with EAX set to 06H, the processor returns information about thermal and power management features. See Table 3-8.

INPUT EAX = 07H: Returns Structured Extended Feature Enumeration Information

When CPUID executes with EAX set to 07H and ECX = 0, the processor returns information about the maximum input value for sub-leaves that contain extended feature flags. See Table 3-8.

When CPUID executes with EAX set to 07H and the input value of ECX is invalid (see leaf 07H entry in Table 3-8), the processor returns 0 in EAX/EBX/ECX/EDX. In subleaf 0, EAX returns the maximum input value of the highest leaf 7 sub-leaf, and EBX, ECX & EDX contain information of extended feature flags.

INPUT EAX = 09H: Returns Direct Cache Access Information

When CPUID executes with EAX set to 09H, the processor returns information about Direct Cache Access capabilities. See Table 3-8.

INPUT EAX = 0AH: Returns Architectural Performance Monitoring Features

When CPUID executes with EAX set to 0AH, the processor returns information about support for architectural performance monitoring capabilities. Architectural performance monitoring is supported if the version ID (see Table 3-8) is greater than Pn 0. See Table 3-8.

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 23, "Introduction to Virtual-Machine Extensions," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.

INPUT EAX = 0BH: Returns Extended Topology Information

CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of Leaf 1FH before using leaf 0BH.

When CPUID executes with EAX set to 0BH, the processor returns information about extended topology enumeration data. Software must detect the presence of CPUID leaf 0BH by verifying (a) the highest leaf index supported by CPUID is >= 0BH, and (b) CPUID.0BH:EBX[15:0] reports a non-zero value. See Table 3-8.

INPUT EAX = 0DH: Returns Processor Extended States Enumeration Information

When CPUID executes with EAX set to 0DH and ECX = 0, the processor returns information about the bit-vector representation of all processor state extensions that are supported in the processor and storage size requirements of the XSAVE/XRSTOR area. See Table 3-8.

When CPUID executes with EAX set to 0DH and ECX = n (n > 1, and is a valid sub-leaf index), the processor returns information about the size and offset of each processor extended state save area within the XSAVE/XRSTOR area. See Table 3-8. Software can use the forward-extendable technique depicted below to query the valid sub-leaves and obtain size and offset information for each processor extended state save area:

```
For i = 2 to 62 // sub-leaf 1 is reserved
    IF (CPUID.(EAX=0DH, ECX=0):VECTOR[i] = 1 ) // VECTOR is the 64-bit value of EDX:EAX
        Execute CPUID.(EAX=0DH, ECX = i) to examine size and offset for sub-leaf i;
    FI;
```

INPUT EAX = 0FH: Returns Intel Resource Director Technology (Intel RDT) Monitoring Enumeration Information

When CPUID executes with EAX set to 0FH and ECX = 0, the processor 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. See Table 3-8.

When CPUID executes with EAX set to 0FH and ECX = n (n >= 1, and is a valid ResID), the processor returns information software can use to program IA32_PQR_ASSOC, IA32_QM_EVTSEL MSRs before reading QoS data from the IA32_QM_CTR MSR.

INPUT EAX = 10H: Returns Intel Resource Director Technology (Intel RDT) Allocation Enumeration Information

When CPUID executes with EAX set to 10H and ECX = 0, the processor 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. See Table 3-8.

When CPUID executes with EAX set to 10H and ECX = n (n >= 1, and is a valid ResID), the processor returns information about available classes of service and range of QoS mask MSRs that software can use to configure each class of services using capability bit masks in the QoS Mask registers, IA32_resourceType_Mask_n.

INPUT EAX = 12H: Returns Intel SGX Enumeration Information

When CPUID executes with EAX set to 12H and ECX = 0H, the processor returns information about Intel SGX capabilities. See Table 3-8.

When CPUID executes with EAX set to 12H and ECX = 1H, the processor returns information about Intel SGX attributes. See Table 3-8.

When CPUID executes with EAX set to 12H and ECX = n (n > 1), the processor returns information about Intel SGX Enclave Page Cache. See Table 3-8.

INPUT EAX = 14H: Returns Intel Processor Trace Enumeration Information

When CPUID executes with EAX set to 14H and ECX = 0H, the processor returns information about Intel Processor Trace extensions. See Table 3-8.

When CPUID executes with EAX set to 14H and ECX = n (n > 0 and less than the number of non-zero bits in CPUID.(EAX=14H, ECX= 0H).EAX), the processor returns information about packet generation in Intel Processor Trace. See Table 3-8.

INPUT EAX = 15H: Returns Time Stamp Counter and Nominal Core Crystal Clock Information

When CPUID executes with EAX set to 15H and ECX = 0H, the processor returns information about Time Stamp Counter and Core Crystal Clock. See Table 3-8.

INPUT EAX = 16H: Returns Processor Frequency Information

When CPUID executes with EAX set to 16H, the processor returns information about Processor Frequency Information. See Table 3-8.

INPUT EAX = 17H: Returns System-On-Chip Information

When CPUID executes with EAX set to 17H, the processor returns information about the System-On-Chip Vendor Attribute Enumeration. See Table 3-8.

INPUT EAX = 18H: Returns Deterministic Address Translation Parameters Information

When CPUID executes with EAX set to 18H, the processor returns information about the Deterministic Address Translation Parameters. See Table 3-8.

INPUT EAX = 19H: Returns Key Locker Information

When CPUID executes with EAX set to 19H, the processor returns information about Key Locker. See Table 3-8.

INPUT EAX = 1AH: Returns Hybrid Information

When CPUID executes with EAX set to 1AH, the processor returns information about hybrid capabilities. See Table 3-8.

INPUT EAX = 1BH: Returns PCONFIG Information

When CPUID executes with EAX set to 1BH, the processor returns information about PCONFIG capabilities. This information is enumerated in sub-leaves selected by the value of ECX (starting with 0).

Each sub-leaf of CPUID function 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, meaning **target identifier**, 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 of the PCONFIG instruction (any value of zero should be ignored). The only target identifier currently defined is 1, indicating MKTME. 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.

INPUT EAX = 1CH: Returns Last Branch Record Information

When CPUID executes with EAX set to 1CH, the processor returns information about LBRs (the architectural feature). See Table 3-8.

INPUT EAX = 1FH: Returns V2 Extended Topology Information

When CPUID executes with EAX set to 1FH, the processor returns information about extended topology enumeration data. Software must detect the presence of CPUID leaf 1FH by verifying (a) the highest leaf index supported by CPUID is >= 1FH, and (b) CPUID.1FH:EBX[15:0] reports a non-zero value. See Table 3-8.

INPUT EAX = 20H: Returns History Reset Information

When CPUID executes with EAX set to 20H, the processor returns information about History Reset. See Table 3-8.

METHODS FOR RETURNING BRANDING INFORMATION

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: "Identification of Earlier IA-32 Processors" in Chapter 20 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

The Processor Brand String Method

Figure 3-9 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.

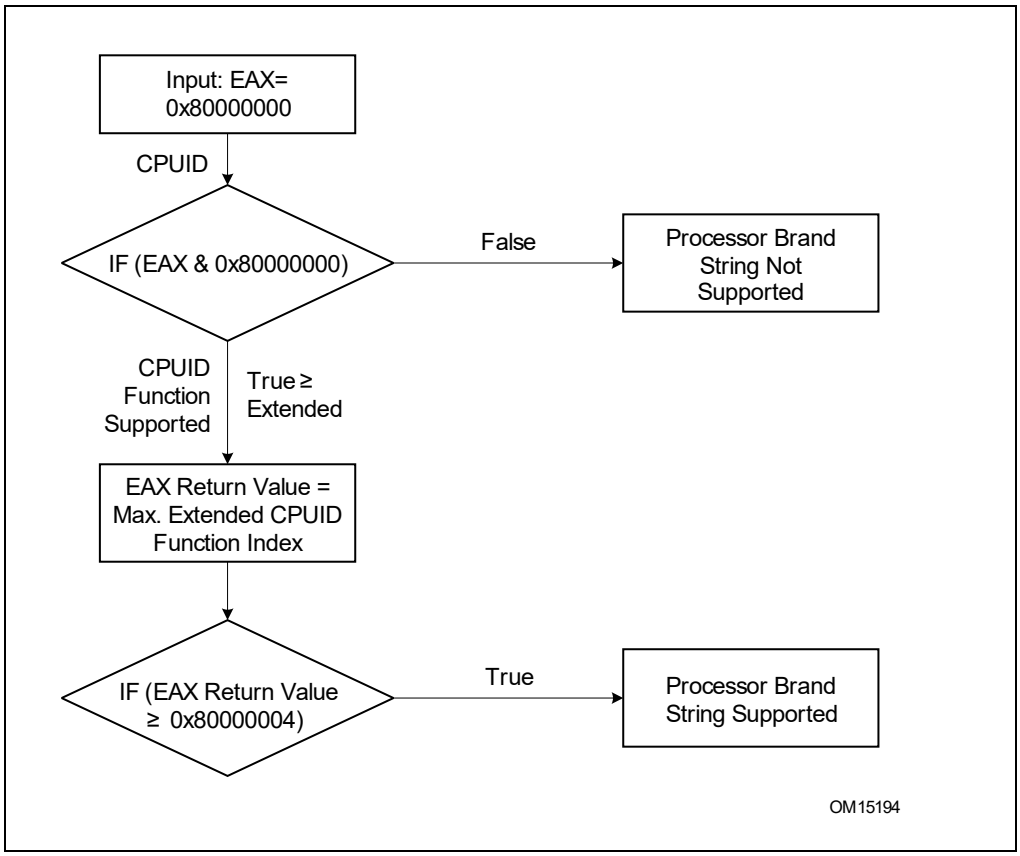


Figure 3-9. Determination of Support for the Processor Brand String

How Brand Strings Work

To use the brand string method, execute CPUID with EAX input of 8000002H through 80000004H. For each input value, CPUID returns 16 ASCII characters using EAX, EBX, ECX, and EDX. The returned string will be NULL-terminated.

Table 3-13 shows the brand string that is returned by the first processor in the Pentium 4 processor family.

Table 3-13. Processor Brand String Returned with Pentium 4 Processor

EAX Input Value	Return Values	ASCII Equivalent
80000002H	EAX = 20202020H EBX = 20202020H ECX = 20202020H EDX = 6E492020H	" " " " " " "nI "
80000003H	EAX = 286C6574H EBX = 50202952H ECX = 69746E65H EDX = 52286D75H	"(let" "P)R" "itne" "R(mu"

Table 3-13. Processor Brand String Returned with Pentium 4 Processor (Contd.)

EAX Input Value	Return Values	ASCII Equivalent
80000004H	EAX = 20342029H	" 4)"
	EBX = 20555043H	" UPC"
	ECX = 30303531H	"0051"
	EDX = 007A484DH	"\0zHM"

Extracting the Processor Frequency from Brand Strings

Figure 3-10 provides an algorithm which software can use to extract the Processor Base frequency from the processor brand string.

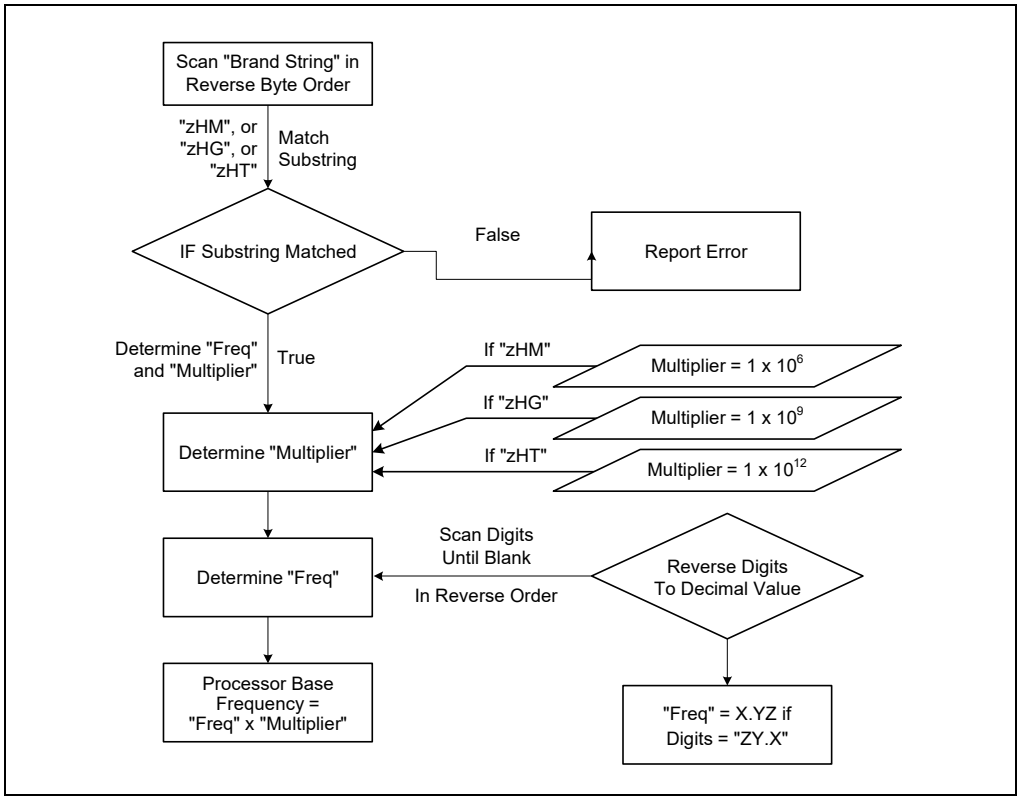


Figure 3-10. Algorithm for Extracting Processor Frequency

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 system software and is accessible from system- and user-level code. In this table, each brand index is associate 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 3-14 shows brand indices that have identification strings associated with them.

Table 3-14. 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(R) Celeron(R) processor[1]
02H	Intel(R) Pentium(R) III processor[1]
03H	Intel(R) Pentium(R) III Xeon(R) processor; If processor signature = 000006B1h, then Intel(R) Celeron(R) processor
04H	Intel(R) Pentium(R) III processor
06H	Mobile Intel(R) Pentium(R) III processor-M
07H	Mobile Intel(R) Celeron(R) processor[1]
08H	Intel(R) Pentium(R) 4 processor
09H	Intel(R) Pentium(R) 4 processor
0AH	Intel(R) Celeron(R) processor[1]
0BH	Intel(R) Xeon(R) processor; If processor signature = 00000F13h, then Intel(R) Xeon(R) processor MP
0CH	Intel(R) Xeon(R) processor MP
0EH	Mobile Intel(R) Pentium(R) 4 processor-M; If processor signature = 00000F13h, then Intel(R) Xeon(R) processor
0FH	Mobile Intel(R) Celeron(R) processor[1]
11H	Mobile Genuine Intel(R) processor
12H	Intel(R) Celeron(R) M processor
13H	Mobile Intel(R) Celeron(R) processor[1]
14H	Intel(R) Celeron(R) processor
15H	Mobile Genuine Intel(R) processor
16H	Intel(R) Pentium(R) M processor
17H	Mobile Intel(R) Celeron(R) processor[1]
18H – 0FFH	RESERVED

NOTES:

1. Indicates versions of these processors that were introduced after the Pentium III

IA-32 Architecture Compatibility

CPUID is not supported in early models of the Intel486 processor or in any IA-32 processor earlier than the Intel486 processor.

Operation

```
IA32_BIOS_SIGN_ID MSR := Update with installed microcode revision number;

CASE (EAX) OF
    EAX = 0:
        EAX := Highest basic function input value understood by CPUID;
        EBX := Vendor identification string;
        EDX := Vendor identification string;
        ECX := Vendor identification string;
    BREAK;
    EAX = 1H:
        EAX[3:0] := Stepping ID;
        EAX[7:4] := Model;
        EAX[11:8] := Family;
```

```
        EAX[13:12] := Processor type;
        EAX[15:14] := Reserved;
        EAX[19:16] := Extended Model;
        EAX[27:20] := Extended Family;
        EAX[31:28] := Reserved;
        EBX[7:0] := Brand Index; (* Reserved if the value is zero. *)
        EBX[15:8] := CLFLUSH Line Size;
        EBX[16:23] := Reserved; (* Number of threads enabled = 2 if MT enable fuse set. *)
        EBX[24:31] := Initial APIC ID;
        ECX := Feature flags; (* See Figure 3-7. *)
        EDX := Feature flags; (* See Figure 3-8. *)
    BREAK;
    EAX = 2H:
        EAX := Cache and TLB information;
        EBX := Cache and TLB information;
        ECX := Cache and TLB information;
        EDX := Cache and TLB information;
    BREAK;
    EAX = 3H:
        EAX := Reserved;
        EBX := Reserved;
        ECX := ProcessorSerialNumber[31:0];
        (* Pentium III processors only, otherwise reserved. *)
        EDX := ProcessorSerialNumber[63:32];
        (* Pentium III processors only, otherwise reserved. *
    BREAK
    EAX = 4H:
        EAX := Deterministic Cache Parameters Leaf; (* See Table 3-8. *)
        EBX := Deterministic Cache Parameters Leaf;
        ECX := Deterministic Cache Parameters Leaf;
        EDX := Deterministic Cache Parameters Leaf;
    BREAK;
    EAX = 5H:
        EAX := MONITOR/MWAIT Leaf; (* See Table 3-8. *)
        EBX := MONITOR/MWAIT Leaf;
        ECX := MONITOR/MWAIT Leaf;
        EDX := MONITOR/MWAIT Leaf;
    BREAK;
    EAX = 6H:
        EAX := Thermal and Power Management Leaf; (* See Table 3-8. *)
        EBX := Thermal and Power Management Leaf;
        ECX := Thermal and Power Management Leaf;
        EDX := Thermal and Power Management Leaf;
    BREAK;
    EAX = 7H:
        EAX := Structured Extended Feature Flags Enumeration Leaf; (* See Table 3-8. *)
        EBX := Structured Extended Feature Flags Enumeration Leaf;
        ECX := Structured Extended Feature Flags Enumeration Leaf;
        EDX := Structured Extended Feature Flags Enumeration Leaf;
    BREAK;
    EAX = 8H:
        EAX := Reserved = 0;
        EBX := Reserved = 0;
        ECX := Reserved = 0;
```

```
        EDX := Reserved = 0;
    BREAK;
    EAX = 9H:
        EAX := Direct Cache Access Information Leaf; (* See Table 3-8. *)
        EBX := Direct Cache Access Information Leaf;
        ECX := Direct Cache Access Information Leaf;
        EDX := Direct Cache Access Information Leaf;
    BREAK;
    EAX = AH:
        EAX := Architectural Performance Monitoring Leaf; (* See Table 3-8. *)
        EBX := Architectural Performance Monitoring Leaf;
        ECX := Architectural Performance Monitoring Leaf;
        EDX := Architectural Performance Monitoring Leaf;
        BREAK
    EAX = BH:
        EAX := Extended Topology Enumeration Leaf; (* See Table 3-8. *)
        EBX := Extended Topology Enumeration Leaf;
        ECX := Extended Topology Enumeration Leaf;
        EDX := Extended Topology Enumeration Leaf;
    BREAK;
    EAX = CH:
        EAX := Reserved = 0;
        EBX := Reserved = 0;
        ECX := Reserved = 0;
        EDX := Reserved = 0;
    BREAK;
    EAX = DH:
        EAX := Processor Extended State Enumeration Leaf; (* See Table 3-8. *)
        EBX := Processor Extended State Enumeration Leaf;
        ECX := Processor Extended State Enumeration Leaf;
        EDX := Processor Extended State Enumeration Leaf;
    BREAK;
    EAX = EH:
        EAX := Reserved = 0;
        EBX := Reserved = 0;
        ECX := Reserved = 0;
        EDX := Reserved = 0;
    BREAK;
    EAX = FH:
        EAX := Intel Resource Director Technology Monitoring Enumeration Leaf; (* See Table 3-8. *)
        EBX := Intel Resource Director Technology Monitoring Enumeration Leaf;
        ECX := Intel Resource Director Technology Monitoring Enumeration Leaf;
        EDX := Intel Resource Director Technology Monitoring Enumeration Leaf;
    BREAK;
    EAX = 10H:
        EAX := Intel Resource Director Technology Allocation Enumeration Leaf; (* See Table 3-8. *)
        EBX := Intel Resource Director Technology Allocation Enumeration Leaf;
        ECX := Intel Resource Director Technology Allocation Enumeration Leaf;
        EDX := Intel Resource Director Technology Allocation Enumeration Leaf;
    BREAK;
    EAX = 12H:
        EAX := Intel SGX Enumeration Leaf; (* See Table 3-8. *)
        EBX := Intel SGX Enumeration Leaf;
        ECX := Intel SGX Enumeration Leaf;
```

```
        EDX := Intel SGX Enumeration Leaf;
    BREAK;
    EAX = 14H:
        EAX := Intel Processor Trace Enumeration Leaf; (* See Table 3-8. *)
        EBX := Intel Processor Trace Enumeration Leaf;
        ECX := Intel Processor Trace Enumeration Leaf;
        EDX := Intel Processor Trace Enumeration Leaf;
    BREAK;
    EAX = 15H:
        EAX := Time Stamp Counter and Nominal Core Crystal Clock Information Leaf; (* See Table 3-8. *)
        EBX := Time Stamp Counter and Nominal Core Crystal Clock Information Leaf;
        ECX := Time Stamp Counter and Nominal Core Crystal Clock Information Leaf;
        EDX := Time Stamp Counter and Nominal Core Crystal Clock Information Leaf;
    BREAK;
    EAX = 16H:
        EAX := Processor Frequency Information Enumeration Leaf; (* See Table 3-8. *)
        EBX := Processor Frequency Information Enumeration Leaf;
        ECX := Processor Frequency Information Enumeration Leaf;
        EDX := Processor Frequency Information Enumeration Leaf;
    BREAK;
    EAX = 17H:
        EAX := System-On-Chip Vendor Attribute Enumeration Leaf; (* See Table 3-8. *)
        EBX := System-On-Chip Vendor Attribute Enumeration Leaf;
        ECX := System-On-Chip Vendor Attribute Enumeration Leaf;
        EDX := System-On-Chip Vendor Attribute Enumeration Leaf;
    BREAK;
    EAX = 18H:
        EAX := Deterministic Address Translation Parameters Enumeration Leaf; (* See Table 3-8. *)
        EBX := Deterministic Address Translation Parameters Enumeration Leaf;
        ECX := Deterministic Address Translation Parameters Enumeration Leaf;
        EDX := Deterministic Address Translation Parameters Enumeration Leaf;
    BREAK;
    EAX = 19H:
        EAX := Key Locker Enumeration Leaf; (* See Table 3-8. *)
        EBX := Key Locker Enumeration Leaf;
        ECX := Key Locker Enumeration Leaf;
        EDX := Key Locker Enumeration Leaf;
    BREAK;
    EAX = 1AH:
        EAX := Hybrid Information Enumeration Leaf; (* See Table 3-8. *)
        EBX := Hybrid Information Enumeration Leaf;
        ECX := Hybrid Information Enumeration Leaf;
        EDX := Hybrid Information Enumeration Leaf;
    BREAK;
    EAX = 1BH:
        EAX := PCONFIG Information Enumeration Leaf; (* See "INPUT EAX = 1BH: Returns PCONFIG Information" on page 3-247. *)
        EBX := PCONFIG Information Enumeration Leaf;
        ECX := PCONFIG Information Enumeration Leaf;
        EDX := PCONFIG Information Enumeration Leaf;
    BREAK;
    EAX = 1CH:
        EAX := Last Branch Record Information Enumeration Leaf; (* See Table 3-8. *)
        EBX := Last Branch Record Information Enumeration Leaf;
        ECX := Last Branch Record Information Enumeration Leaf;
```

```
        EDX := Last Branch Record Information Enumeration Leaf;
    BREAK;
    EAX = 1FH:
        EAX := V2 Extended Topology Enumeration Leaf; (* See Table 3-8. *)
        EBX := V2 Extended Topology Enumeration Leaf;
        ECX := V2 Extended Topology Enumeration Leaf;
        EDX := V2 Extended Topology Enumeration Leaf;
    BREAK;
    EAX = 20H:
        EAX := Processor History Reset Sub-leaf; (* See Table 3-8. *)
        EBX := Processor History Reset Sub-leaf;
        ECX := Processor History Reset Sub-leaf;
        EDX := Processor History Reset Sub-leaf;
    BREAK;
    EAX = 80000000H:
        EAX := Highest extended function input value understood by CPUID;
        EBX := Reserved;
        ECX := Reserved;
        EDX := Reserved;
    BREAK;
    EAX = 80000001H:
        EAX := Reserved;
        EBX := Reserved;
        ECX := Extended Feature Bits (* See Table 3-8.*);
        EDX := Extended Feature Bits (* See Table 3-8. *);
    BREAK;
    EAX = 80000002H:
        EAX := Processor Brand String;
        EBX := Processor Brand String, continued;
        ECX := Processor Brand String, continued;
        EDX := Processor Brand String, continued;
    BREAK;
    EAX = 80000003H:
        EAX := Processor Brand String, continued;
        EBX := Processor Brand String, continued;
        ECX := Processor Brand String, continued;
        EDX := Processor Brand String, continued;
    BREAK;
    EAX = 80000004H:
        EAX := Processor Brand String, continued;
        EBX := Processor Brand String, continued;
        ECX := Processor Brand String, continued;
        EDX := Processor Brand String, continued;
    BREAK;
    EAX = 80000005H:
        EAX := Reserved = 0;
        EBX := Reserved = 0;
        ECX := Reserved = 0;
        EDX := Reserved = 0;
    BREAK;
    EAX = 80000006H:
        EAX := Reserved = 0;
        EBX := Reserved = 0;
        ECX := Cache information;
```

```
        EDX := Reserved = 0;
    BREAK;
    EAX = 80000007H:
        EAX := Reserved = 0;
        EBX := Reserved = 0;
        ECX := Reserved = 0;
        EDX := Reserved = Misc Feature Flags;
    BREAK;
    EAX = 80000008H:
        EAX := Reserved = Physical Address Size Information;
        EBX := Reserved = Virtual Address Size Information;
        ECX := Reserved = 0;
        EDX := Reserved = 0;
    BREAK;
    EAX >= 40000000H and EAX <= 4FFFFFFFH:
    DEFAULT: (* EAX = Value outside of recognized range for CPUID. *)
        (* If the highest basic information leaf data depend on ECX input value, ECX is honored.*)
        EAX := Reserved; (* Information returned for highest basic information leaf. *)
        EBX := Reserved; (* Information returned for highest basic information leaf. *)
        ECX := Reserved; (* Information returned for highest basic information leaf. *)
        EDX := Reserved; (* Information returned for highest basic information leaf. *)
    BREAK;
ESAC;
```

Flags Affected

None.

Exceptions (All Operating Modes)

#UD	If the LOCK prefix is used. In earlier IA-32 processors that do not support the CPUID instruction, execution of the instruction results in an invalid opcode (#UD) exception being generated.

ENTER—Make Stack Frame for Procedure Parameters

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
C8 *iw* 00	ENTER *imm16*, 0	II	Valid	Valid	Create a stack frame for a procedure.
C8 *iw* 01	ENTER *imm16*,1	II	Valid	Valid	Create a stack frame with a nested pointer for a procedure.
C8 *iw* ib	ENTER *imm16, imm8*	II	Valid	Valid	Create a stack frame with nested pointers for a procedure.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
II	iw	imm8	NA	NA

Description

Creates a stack frame (comprising of space for dynamic storage and 1-32 frame pointer storage) for a procedure. The first operand (imm16) specifies the size of the dynamic storage in the stack frame (that is, the number of bytes of dynamically allocated on the stack for the procedure). The second operand (imm8) gives the lexical nesting level (0 to 31) of the procedure. The nesting level (imm8 mod 32) and the OperandSize attribute determine the size in bytes of the storage space for frame pointers.

The nesting level determines the number of frame pointers that are copied into the "display area" of the new stack frame from the preceding frame. The default size of the frame pointer is the StackAddrSize attribute, but can be overridden using the 66H prefix. Thus, the OperandSize attribute determines the size of each frame pointer that will be copied into the stack frame and the data being transferred from SP/ESP/RSP register into the BP/EBP/RBP register.

The ENTER and companion LEAVE instructions are provided to support block structured languages. The ENTER instruction (when used) is typically the first instruction in a procedure and is used to set up a new stack frame for a procedure. The LEAVE instruction is then used at the end of the procedure (just before the RET instruction) to release the stack frame.

If the nesting level is 0, the processor pushes the frame pointer from the BP/EBP/RBP register onto the stack, copies the current stack pointer from the SP/ESP/RSP register into the BP/EBP/RBP register, and loads the SP/ESP/RSP register with the current stack-pointer value minus the value in the size operand. For nesting levels of 1 or greater, the processor pushes additional frame pointers on the stack before adjusting the stack pointer. These additional frame pointers provide the called procedure with access points to other nested frames on the stack. See "Procedure Calls for Block-Structured Languages" in Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for more information about the actions of the ENTER instruction.

The ENTER instruction causes a page fault whenever a write using the final value of the stack pointer (within the current stack segment) would do so.

In 64-bit mode, default operation size is 64 bits; 32-bit operation size cannot be encoded. Use of 66H prefix changes frame pointer operand size to 16 bits.

When the 66H prefix is used and causing the OperandSize attribute to be less than the StackAddrSize, software is responsible for the following:

- The companion LEAVE instruction must also use the 66H prefix,
- The value in the RBP/EBP register prior to executing "66H ENTER" must be within the same 16KByte region of the current stack pointer (RSP/ESP), such that the value of RBP/EBP after "66H ENTER" remains a valid address in the stack. This ensures "66H LEAVE" can restore 16-bits of data from the stack.

Operation

```
AllocSize := imm16;
NestingLevel := imm8 MOD 32;
IF (OperandSize = 64)
    THEN
        Push(RBP); (* RSP decrements by 8 *)
        FrameTemp := RSP;
    ELSE IF OperandSize = 32
        THEN
            Push(EBP); (* (E)SP decrements by 4 *)
            FrameTemp := ESP; FI;
    ELSE (* OperandSize = 16 *)
            Push(BP); (* RSP or (E)SP decrements by 2 *)
            FrameTemp := SP;
FI;

IF NestingLevel = 0
    THEN GOTO CONTINUE;
FI;

IF (NestingLevel > 1)
    THEN FOR i := 1 to (NestingLevel - 1)
        DO
            IF (OperandSize = 64)
                THEN
                    RBP := RBP - 8;
                    Push([RBP]); (* Quadword push *)
                ELSE IF OperandSize = 32
                    THEN
                        IF StackSize = 32
                            EBP := EBP - 4;
                            Push([EBP]); (* Doubleword push *)
                        ELSE (* StackSize = 16 *)
                            BP := BP - 4;
                            Push([BP]); (* Doubleword push *)
                        FI;
                    FI;
                ELSE (* OperandSize = 16 *)
                    IF StackSize = 64
                        THEN
                            RBP := RBP - 2;
                            Push([RBP]); (* Word push *)
                    ELSE IF StackSize = 32
                        THEN
                            EBP := EBP - 2;
                            Push([EBP]); (* Word push *)
                        ELSE (* StackSize = 16 *)
                            BP := BP - 2;
                            Push([BP]); (* Word push *)
                    FI;
                FI;
    OD;
FI;
```

```
IF (OperandSize = 64) (* nestinglevel 1 *)
    THEN
        Push(FrameTemp); (* Quadword push and RSP decrements by 8 *)
    ELSE IF OperandSize = 32
        THEN
            Push(FrameTemp); FI; (* Doubleword push and (E)SP decrements by 4 *)
    ELSE (* OperandSize = 16 *)
            Push(FrameTemp); (* Word push and RSP|ESP|SP decrements by 2 *)
FI;

CONTINUE:
IF 64-Bit Mode (StackSize = 64)
    THEN
            RBP := FrameTemp;
            RSP := RSP – AllocSize;
    ELSE IF OperandSize = 32
        THEN
            EBP := FrameTemp;
            ESP := ESP – AllocSize; FI;
    ELSE (* OperandSize = 16 *)
            BP := FrameTemp[15:1]; (* Bits 16 and above of applicable RBP/EBP are unmodified *)
            SP := SP – AllocSize;
FI;

END;
```

Flags Affected

None.

Protected Mode Exceptions

#SS(0)	If the new value of the SP or ESP register is outside the stack segment limit.
#PF(fault-code)	If a page fault occurs or if a write using the final value of the stack pointer (within the current stack segment) would cause a page fault.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#SS	If the new value of the SP or ESP register is outside the stack segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#SS(0)	If the new value of the SP or ESP register is outside the stack segment limit.
#PF(fault-code)	If a page fault occurs or if a write using the final value of the stack pointer (within the current stack segment) would cause a page fault.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If the stack address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs or if a write using the final value of the stack pointer (within the current stack segment) would cause a page fault.
#UD	If the LOCK prefix is used.

HRESET — History Reset

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 3A F0 C0 /ib HRESET imm8, <EAX>	A	V/V	HRESET	Processor history reset request. Controlled by the EAX implicit operand.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	ModRM:r/m (r)	NA	NA	NA

Description

Requests the processor to selectively reset selected components of hardware history maintained by the current logical processor. HRESET operation is controlled by the implicit EAX operand. The value of the explicit imm8 operand is ignored. This instruction can only be executed at privilege level 0.

The HRESET instruction can be used to request reset of multiple components of hardware history. Prior to the execution of HRESET, the system software must take the following steps:

1. Enumerate the HRESET capabilities via CPUID.20H.0H:EBX, which indicates what components of hardware history can be reset.
2. Only the bits enumerated by CPUID.20H.0H:EBX can be set in the IA32_HRESET_ENABLE MSR.

HRESET causes a general-protection exception (#GP) if EAX sets any bits that are not set in the IA32_HRESET_ENABLE MSR.

Any attempt to execute the HRESET instruction inside a transactional region will result in a transaction abort.

Operation

```
IF EAX = 0
   THEN NOP
   ELSE
      FOREACH i such that EAX[i] = 1
         Reset prediction history for feature i
FI
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If CPL > 0 or (EAX AND NOT IA32_HRESET_ENABLE) ≠0.
#UD	If CPUID.07H.01H:EAX.HRESET[bit 22] = 0.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

#GP(0)	HRESET 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*/INTO/INT3/INT1—Call to Interrupt Procedure

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
CC	INT3	ZO	Valid	Valid	Generate breakpoint trap.
CD *ib*	INT *imm8*	I	Valid	Valid	Generate software interrupt with vector specified by immediate byte.
CE	INTO	ZO	Invalid	Valid	Generate overflow trap if overflow flag is 1.
F1	INT1	ZO	Valid	Valid	Generate debug trap.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
ZO	NA	NA	NA	NA
I	imm8	NA	NA	NA

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.

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.

(These features do not pertain to CD03, the "normal" 2-byte opcode for INT 3. 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 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

1. The mnemonic ICEBP has also been used for the instruction with opcode F1.

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 3-52. 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.

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 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;
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 ((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;
    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 *)
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;
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 & FFFFFFFFFFFFFFF0H;
            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
                (* These stack pushes should not cause faults, VM exits, data breakpoints *)
                (* Such events will apply to the earlier accesses to the token, which is in the same 32-byte region *)
                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_PLi_SSP (* where i = new code-segment DPL *)
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(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
            (* These stack pushes should not cause faults, VM exits, or data breakpoints *)
            (* Such events will apply to the earlier accesses to the token, which is in the same 32-byte region *)
            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;
(* 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;
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 & FFFFFFFFFFFFFFF0H;
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 *)
                    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 & 0xFFFFFFFFFFFFFFF8H;
            (* These stack pushes should not cause faults, VM exits, or data breakpoints, if CHECK_SS_TOKEN is 1 *)
            (* Such events will apply to the earlier accesses to the token, which is in the same 32-byte region *)
            (* 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

Same exceptions as in protected mode.

64-Bit 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.
	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 a push of the old EFLAGS, CS selector, EIP, or error code is in non-canonical space with no stack switch.
	If a push of the old SS selector, ESP, EFLAGS, CS selector, EIP, or error code is in non-canonical space on a stack switch (either CPL change or no-CPL with IST).
#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.

8. Updates to Chapter 4, Volume 2B

Change bars and green 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 the SERIALIZE instruction.

SERIALIZE — Serialize Instruction Execution

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 E8 SERIALIZE	ZO	V/V	SERIALIZE	Serialize instruction fetch and execution.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
ZO	NA	NA	NA	NA	NA

Description

Serializes instruction execution. Before the next instruction is fetched and executed, the SERIALIZE instruction ensures that all modifications to flags, registers, and memory by previous instructions are completed, draining all buffered writes to memory. This instruction is also a serializing instruction as defined in the section "Serializing Instructions" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

SERIALIZE does not modify registers, arithmetic flags, or memory.

Operation

```
Wait_On_Fetch_And_Execution_Of_Next_Instruction_Until(preceding_instructions_complete_and_preceding_stores_globally_visible);
```

Intel C/C++ Compiler Intrinsic Equivalent

SERIALIZE void _serialize(void);

SIMD Floating-Point Exceptions

None.

Other Exceptions

#UD	If the LOCK prefix is used.
	If CPUID.07H.0H:EDX.SERIALIZE[bit 14] = 0.

9. Updates to Chapter 5, Volume 2C

Change bars and green text show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C:* Instruction Set Reference, V-Z.

Changes to this chapter:

Updates to the following instructions: VGETEXPPD, VGETEXPPS, VGETMANTPD, VGETMANTPS, VGETMANTSD, VGETMANTSS, WRSSD/WRSSQ, and WRUSSD/WRUSSQ.

Added AVX-VNNI versions of the following instructions: VPDPBUSD, VPDPBUSDS, VPDPWSSD, and VPDPWSSDS.

VGETEXPPD—Convert Exponents of Packed DP FP Values to DP FP Values

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.128.66.0F38.W1 42 /r VGETEXPPD xmm1 {k1}{z}, xmm2/m128/m64bcst	A	V/V	AVX512VL AVX512F	Convert the exponent of packed double-precision floating-point values in the source operand to DP FP results representing unbiased integer exponents and stores the results in the destination register.
EVEX.256.66.0F38.W1 42 /r VGETEXPPD ymm1 {k1}{z}, ymm2/m256/m64bcst	A	V/V	AVX512VL AVX512F	Convert the exponent of packed double-precision floating-point values in the source operand to DP FP results representing unbiased integer exponents and stores the results in the destination register.
EVEX.512.66.0F38.W1 42 /r VGETEXPPD zmm1 {k1}{z}, zmm2/m512/m64bcst{sae}	A	V/V	AVX512F	Convert the exponent of packed double-precision floating-point values in the source operand to DP FP results representing unbiased integer exponents and stores the results in the destination under 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)	NA	NA

Description

Extracts the biased exponents from the normalized DP FP representation of each qword data element of the source operand (the second operand) as unbiased signed integer value, or convert the denormal representation of input data to unbiased negative integer values. Each integer value of the unbiased exponent is converted to double-precision FP value and written to the corresponding qword elements of the destination operand (the first operand) as DP FP numbers.

The destination operand is a ZMM/YMM/XMM register and updated under the writemask. The source operand can be a ZMM/YMM/XMM register, a 512/256/128-bit memory location, or a 512/256/128-bit vector broadcasted from a 64-bit memory location.

EVEX.vvvv is reserved and must be 1111b, otherwise instructions will #UD.

Each GETEXP operation converts the exponent value into a FP number (permitting input value in denormal representation). Special cases of input values are listed in Table 5-5.

The formula is:

$GETEXP(x) = floor(\log_2(|x|))$

Notation **floor(x)** stands for the greatest integer not exceeding real number x.

Table 5-5. VGETEXPPD/SD Special Cases

Input Operand	Result	Comments
src1 = NaN	QNaN(src1)	If (SRC = SNaN) then #IE If (SRC = denormal) then #DE
0 < |src1| < INF	$floor(\log_2(\|src1\|))$	
| src1| = +INF	+INF	
| src1| = 0	-INF	

Operation

```
NormalizeExpTinyDPFP(SRC[63:0])
{
    // Jbit is the hidden integral bit of a FP number. In case of denormal number it has the value of ZERO.
    Src.Jbit := 0;
    Dst.exp := 1;
    Dst.fraction := SRC[51:0];
    WHILE(Src.Jbit = 0)
    {
        Src.Jbit := Dst.fraction[51];              // Get the fraction MSB
        Dst.fraction := Dst.fraction << 1 ;        // One bit shift left
        Dst.exp-- ;                // Decrement the exponent
    }
    Dst.fraction := 0;             // zero out fraction bits
    Dst.sign := 1;                 // Return negative sign
    TMP[63:0] := MXCSR.DAZ? 0 : (Dst.sign << 63) OR (Dst.exp << 52) OR (Dst.fraction) ;
    Return (TMP[63:0]);
}

ConvertExpDPFP(SRC[63:0])
{
    Src.sign := 0;                 // Zero out sign bit
    Src.exp := SRC[62:52];
    Src.fraction := SRC[51:0];
    // Check for NaN
    IF (SRC = NaN)
    {
        IF ( SRC = SNAN ) SET IE;
        Return QNAN(SRC);
    }
    // Check for +INF
    IF (Src = +INF) RETURN (Src);

    // check if zero operand
    IF ((Src.exp = 0) AND ((Src.fraction = 0) OR (MXCSR.DAZ = 1))) Return (-INF);
    }
    ELSE        // check if denormal operand (notice that MXCSR.DAZ = 0)
    {
        IF ((Src.exp = 0) AND (Src.fraction != 0))
        {
            TMP[63:0] := NormalizeExpTinyDPFP(SRC[63:0]) ;     // Get Normalized Exponent
            Set #DE
        }
        ELSE        // exponent value is correct
        {
            TMP[63:0] := (Src.sign << 63) OR (Src.exp << 52) OR (Src.fraction) ;
        }
        TMP := SAR(TMP, 52) ;          // Shift Arithmetic Right
        TMP := TMP – 1023;             // Subtract Bias
        Return Cvtl2D(TMP);            // Convert INT to Double-Precision FP number
    }
}
```

VGETEXPPD (EVEX encoded versions)

```
(KL, VL) = (2, 128), (4, 256), (8, 512)
FOR j := 0 TO KL-1
    i := j * 64
    IF k1[j] OR *no writemask*
        THEN
            IF (EVEX.b = 1) AND (SRC *is memory*)
                THEN
                    DEST[i+63:i] :=
            ConvertExpDPFP(SRC[63:0])
                ELSE
                    DEST[i+63:i] :=
            ConvertExpDPFP(SRC[i+63:i])
            FI;
        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
```

Intel C/C++ Compiler Intrinsic Equivalent

```
VGETEXPPD __m512d _mm512_getexp_pd(__m512d a);
VGETEXPPD __m512d _mm512_mask_getexp_pd(__m512d s, __mmask8 k, __m512d a);
VGETEXPPD __m512d _mm512_maskz_getexp_pd( __mmask8 k, __m512d a);
VGETEXPPD __m512d _mm512_getexp_round_pd(__m512d a, int sae);
VGETEXPPD __m512d _mm512_mask_getexp_round_pd(__m512d s, __mmask8 k, __m512d a, int sae);
VGETEXPPD __m512d _mm512_maskz_getexp_round_pd( __mmask8 k, __m512d a, int sae);
VGETEXPPD __m256d _mm256_getexp_pd(__m256d a);
VGETEXPPD __m256d _mm256_mask_getexp_pd(__m256d s, __mmask8 k, __m256d a);
VGETEXPPD __m256d _mm256_maskz_getexp_pd( __mmask8 k, __m256d a);
VGETEXPPD __m128d _mm_getexp_pd(__m128d a);
VGETEXPPD __m128d _mm_mask_getexp_pd(__m128d s, __mmask8 k, __m128d a);
VGETEXPPD __m128d _mm_maskz_getexp_pd( __mmask8 k, __m128d a);
```

SIMD Floating-Point Exceptions

Invalid, Denormal

Other Exceptions

See Table 2-46, "Type E2 Class Exception Conditions"; additionally:

#UD	If EVEX.vvvv != 1111B.

VGETEXPPS—Convert Exponents of Packed SP FP Values to SP FP Values

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.128.66.0F38.W0 42 /r VGETEXPPS xmm1 {k1}{z}, xmm2/m128/m32bcst	A	V/V	AVX512VL AVX512F	Convert the exponent of packed single-precision floating-point values in the source operand to SP FP results representing unbiased integer exponents and stores the results in the destination register.
EVEX.256.66.0F38.W0 42 /r VGETEXPPS ymm1 {k1}{z}, ymm2/m256/m32bcst	A	V/V	AVX512VL AVX512F	Convert the exponent of packed single-precision floating-point values in the source operand to SP FP results representing unbiased integer exponents and stores the results in the destination register.
EVEX.512.66.0F38.W0 42 /r VGETEXPPS zmm1 {k1}{z}, zmm2/m512/m32bcst{sae}	A	V/V	AVX512F	Convert the exponent of packed single-precision floating-point values in the source operand to SP FP results representing unbiased integer exponents and stores the results in the destination register.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	ModRM:r/m (r)	NA	NA

Description

Extracts the biased exponents from the normalized SP FP representation of each dword element of the source operand (the second operand) as unbiased signed integer value, or convert the denormal representation of input data to unbiased negative integer values. Each integer value of the unbiased exponent is converted to single-precision FP value and written to the corresponding dword elements of the destination operand (the first operand) as SP FP numbers.

The destination operand is a ZMM/YMM/XMM register and updated under the writemask. The source operand can be 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.

EVEX.vvvv is reserved and must be 1111b, otherwise instructions will #UD.

Each GETEXP operation converts the exponent value into a FP number (permitting input value in denormal representation). Special cases of input values are listed in Table 5-6.

The formula is:

GETEXP(x) = floor($\log_2$(|x|))

Notation **floor(x)** stands for maximal integer not exceeding real number x.

Software usage of VGETEXPxx and VGETMANTxx instructions generally involve a combination of GETEXP operation and GETMANT operation (see VGETMANTPD). Thus VGETEXPxx instruction do not require software to handle SIMD FP exceptions.

Table 5-6. VGETEXPPS/SS Special Cases

Input Operand	Result	Comments
src1 = NaN	QNaN(src1)	If (SRC = SNaN) then #IE If (SRC = denormal) then #DE
0 < |src1| < INF	floor($\log_2$(|src1|))	
| src1| = +INF	+INF	
| src1| = 0	-INF	

Figure 5-14 illustrates the VGETEXPPS functionality on input values with normalized representation.

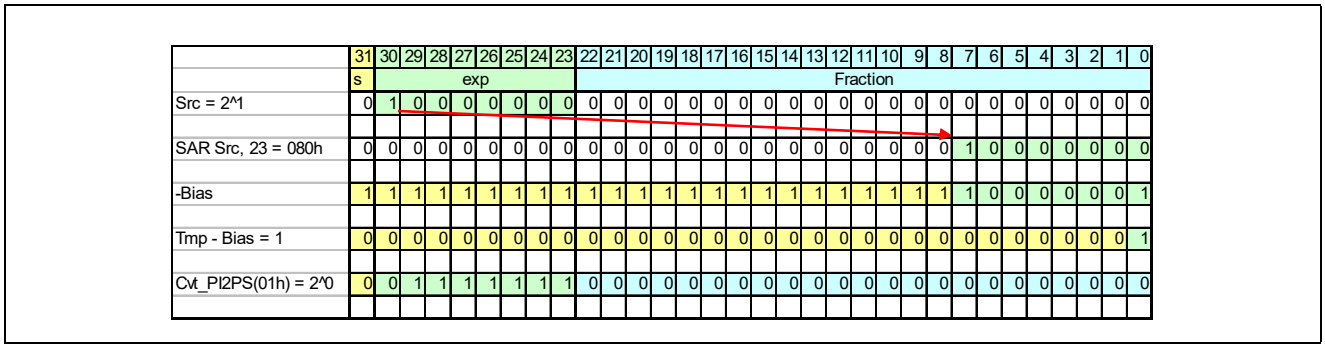


Figure 5-14. VGETEXPPS Functionality On Normal Input values

Operation

```
NormalizeExpTinySPFP(SRC[31:0])
{
    // Jbit is the hidden integral bit of a FP number. In case of denormal number it has the value of ZERO.
    Src.Jbit := 0;
    Dst.exp := 1;
    Dst.fraction := SRC[22:0];
    WHILE(Src.Jbit = 0)
    {
        Src.Jbit := Dst.fraction[22];          // Get the fraction MSB
        Dst.fraction := Dst.fraction << 1 ;    // One bit shift left
        Dst.exp-- ;                            // Decrement the exponent
    }
    Dst.fraction := 0;                         // zero out fraction bits
    Dst.sign := 1;                             // Return negative sign
    TMP[31:0] := MXCSR.DAZ? 0 : (Dst.sign << 31) OR (Dst.exp << 23) OR (Dst.fraction) ;
    Return (TMP[31:0]);
}
ConvertExpSPFP(SRC[31:0])
{
    Src.sign := 0;                             // Zero out sign bit
    Src.exp := SRC[30:23];
    Src.fraction := SRC[22:0];
    // Check for NaN
    IF (SRC = NaN)
    {
        IF ( SRC = SNAN ) SET IE;
        Return QNAN(SRC);
    }
    // Check for +INF
    IF (Src = +INF) RETURN (Src);

    // check if zero operand
    IF ((Src.exp = 0) AND ((Src.fraction = 0) OR (MXCSR.DAZ = 1))) Return (-INF);
    }
    ELSE          // check if denormal operand (notice that MXCSR.DAZ = 0)
    {
```

```
        IF ((Src.exp = 0) AND (Src.fraction != 0))
        {
            TMP[31:0] := NormalizeExpTinySPFP(SRC[31:0]) ;      // Get Normalized Exponent
            Set #DE
        }
        ELSE            // exponent value is correct
        {
            TMP[31:0] := (Src.sign << 31) OR (Src.exp << 23) OR (Src.fraction) ;
        }
        TMP := SAR(TMP, 23) ;               // Shift Arithmetic Right
        TMP := TMP – 127;                   // Subtract Bias
        Return Cvtl2S(TMP);                 // Convert INT to Single-Precision FP number
    }
}
```

VGETEXPPS (EVEX encoded versions)

```
(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) AND (SRC *is memory*)
                THEN
                    DEST[i+31:i] :=
            ConvertExpSPFP(SRC[31:0])
                ELSE
                    DEST[i+31:i] :=
            ConvertExpSPFP(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;
ENDFOR
DEST[MAXVL-1:VL] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

VGETEXPPS __m512 _mm512_getexp_ps(__m512 a);
VGETEXPPS __m512 _mm512_mask_getexp_ps(__m512 s, __mmask16 k, __m512 a);
VGETEXPPS __m512 _mm512_maskz_getexp_ps(__mmask16 k, __m512 a);
VGETEXPPS __m512 _mm512_getexp_round_ps(__m512 a, int sae);
VGETEXPPS __m512 _mm512_mask_getexp_round_ps(__m512 s, __mmask16 k, __m512 a, int sae);
VGETEXPPS __m512 _mm512_maskz_getexp_round_ps(__mmask16 k, __m512 a, int sae);
VGETEXPPS __m256 _mm256_getexp_ps(__m256 a);
VGETEXPPS __m256 _mm256_mask_getexp_ps(__m256 s, __mmask8 k, __m256 a);
VGETEXPPS __m256 _mm256_maskz_getexp_ps(__mmask8 k, __m256 a);
VGETEXPPS __m128 _mm_getexp_ps(__m128 a);
VGETEXPPS __m128 _mm_mask_getexp_ps(__m128 s, __mmask8 k, __m128 a);
VGETEXPPS __m128 _mm_maskz_getexp_ps(__mmask8 k, __m128 a);

SIMD Floating-Point Exceptions

Invalid, Denormal

Other Exceptions

See Table 2-46, “Type E2 Class Exception Conditions”; additionally:

#UD	If EVEX.vvvv != 1111B.

VGETMANTPD—Extract Float64 Vector of Normalized Mantissas from Float64 Vector

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.128.66.0F3A.W1 26 /r ib VGETMANTPD xmm1 {k1}{z}, xmm2/m128/m64bcst, imm8	A	V/V	AVX512VL AVX512F	Get Normalized Mantissa from float64 vector xmm2/m128/m64bcst and store the result in xmm1, using *imm*8 for sign control and mantissa interval normalization, under writemask.
EVEX.256.66.0F3A.W1 26 /r ib VGETMANTPD ymm1 {k1}{z}, ymm2/m256/m64bcst, imm8	A	V/V	AVX512VL AVX512F	Get Normalized Mantissa from float64 vector ymm2/m256/m64bcst and store the result in ymm1, using *imm*8 for sign control and mantissa interval normalization, under writemask.
EVEX.512.66.0F3A.W1 26 /r ib VGETMANTPD zmm1 {k1}{z}, zmm2/m512/m64bcst{sae}, imm8	A	V/V	AVX512F	Get Normalized Mantissa from float64 vector zmm2/m512/m64bcst and store the result in zmm1, using *imm*8 for sign control and mantissa interval normalization, under writemask.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	ModRM:r/m (r)	Imm8	NA

Description

Convert double-precision floating values in the source operand (the second operand) to DP FP values with the mantissa normalization and sign control specified by the imm8 byte, see Figure 5-15. The converted results are written to the destination operand (the first operand) using writemask k1. The normalized mantissa is specified by interv (imm8[1:0]) and the sign control (sc) is specified by bits 3:2 of the immediate byte.

The destination operand is a ZMM/YMM/XMM register updated under the writemask. The source operand can be a ZMM/YMM/XMM register, a 512/256/128-bit memory location, or a 512/256/128-bit vector broadcasted from a 64-bit memory location.

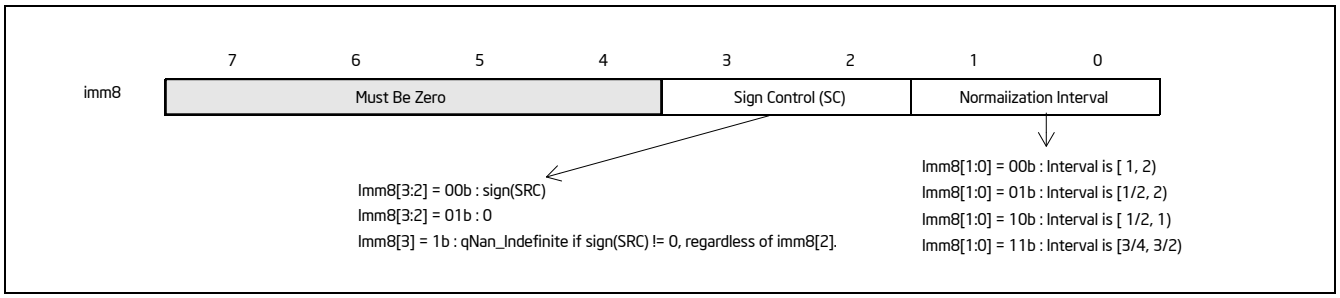


Figure 5-15. Imm8 Controls for VGETMANTPD/SD/PS/SS

For each input DP FP value x, The conversion operation is:

$$GetMant(x) = \pm 2^{k}|x.significand|$$

where:

$$1 <= |x.significand| < 2$$

Unbiased exponent k can be either 0 or -1, depending on the interval range defined by interv, the range of the significand and whether the exponent of the source is even or odd. The sign of the final result is determined by sc and the source sign. The encoded value of imm8[1:0] and sign control are shown in Figure 5-15.

Each converted DP FP result is encoded according to the sign control, the unbiased exponent k (adding bias) and a mantissa normalized to the range specified by interv.

The GetMant() function follows Table 5-7 when dealing with floating-point special numbers.

This instruction is writemasked, so only those elements with the corresponding bit set in vector mask register k1 are computed and stored into the destination. Elements in zmm1 with the corresponding bit clear in k1 retain their previous values.

Note: EVEX.vvvv is reserved and must be 1111b; otherwise instructions will #UD.

Table 5-7. GetMant() Special Float Values Behavior

Input	Result	Exceptions / Comments
NaN	QNaN(SRC)	Ignore *interv* If (SRC = SNaN) then #IE
+∞	1.0	Ignore *interv*
+0	1.0	Ignore *interv*
-0	IF (SC[0]) THEN +1.0 ELSE -1.0	Ignore *interv*

-∞	IF (SC[1]) THEN {QNaN_Indefinite}
	ELSE {
	IF (SC[0]) THEN +1.0
	ELSE -1.0
		Ignore *interv*
		If (SC[1]) then #IE

negative	SC[1] ? QNaN_Indefinite : Getmant(SRC)[1]	If (SC[1]) then #IE

NOTES:

1. In case SC[1]==0, the sign of Getmant(SRC) is declared according to SC[0].

Operation

```
def getmant_fp64(src, sign_control, normalization_interval):
    bias := 1023
    dst.sign := sign_control[0] ? 0 : src.sign
    signed_one := sign_control[0] ? +1.0 : -1.0
    dst.exp := src.exp
    dst.fraction := src.fraction
    zero := (dst.exp = 0) and ((dst.fraction = 0) or (MXCSR.DAZ=1))
    denormal := (dst.exp = 0) and (dst.fraction != 0) and (MXCSR.DAZ=0)
    infinity := (dst.exp = 0x7FF) and (dst.fraction = 0)
    nan := (dst.exp = 0x7FF) and (dst.fraction != 0)
    src_signaling := src.fraction[51]
    snan := nan and (src_signaling = 0)
    positive := (src.sign = 0)
    negative := (src.sign = 1)
    if nan:
        if snan:
            MXCSR.IE := 1
        return qnan(src)

    if positive and (zero or infinity):
        return 1.0
    if negative:
        if zero:
            return signed_one
        if infinity:
```

```
                if sign_control[1]:
                    MXCSR.IE := 1
                    return QNaN_Indefinite
                return signed_one
            if sign_control[1]:
                MXCSR.IE := 1
                return QNaN_Indefinite

    if denormal:
        jbit := 0
        dst.exp := bias
        while jbit = 0:
            jbit := dst.fraction[51]
            dst.fraction := dst.fraction << 1
            dst.exp : = dst.exp - 1
        MXCSR.DE := 1

    unbiased_exp := dst.exp - bias
    odd_exp := unbiased_exp[0]
    signaling_bit := dst.fraction[51]
    if normalization_interval = 0b00:
        dst.exp := bias
    else if normalization_interval = 0b01:
        dst.exp := odd_exp ? bias-1 : bias
    else if normalization_interval = 0b10:
        dst.exp := bias-1
    else if normalization_interval = 0b11:
        dst.exp := signaling_bit ? bias-1 : bias
    return dst
```

VGETMANTPD (EVEX encoded versions)

```
VGETMANTPD dest{k1}, src, imm8
VL = 128, 256, or 512
KL := VL / 64
sign_control := imm8[3:2]
normalization_interval := imm8[1:0]

FOR i := 0 to KL-1:
    IF k1[i] or *no writemask*:
        IF SRC is memory and (EVEX.b = 1):
            tsrc := src.double[0]
        ELSE:
            tsrc := src.double[i]
        DEST.double[i] := getmant_fp64(tsrc, sign_control, normalization_interval)
    ELSE IF *zeroing*:
        DEST.double[i] := 0
    //else DEST.double[i] remains unchanged

DEST[MAX_VL-1:VL] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

VGETMANTPD __m512d _mm512_getmant_pd(__m512d a, enum intv, enum sgn);
VGETMANTPD __m512d _mm512_mask_getmant_pd(__m512d s, __mmask8 k, __m512d a, enum intv, enum sgn);
VGETMANTPD __m512d _mm512_maskz_getmant_pd(__mmask8 k, __m512d a, enum intv, enum sgn);
VGETMANTPD __m512d _mm512_getmant_round_pd(__m512d a, enum intv, enum sgn, int r);
VGETMANTPD __m512d _mm512_mask_getmant_round_pd(__m512d s, __mmask8 k, __m512d a, enum intv, enum sgn, int r);
VGETMANTPD __m512d _mm512_maskz_getmant_round_pd(__mmask8 k, __m512d a, enum intv, enum sgn, int r);
VGETMANTPD __m256d _mm256_getmant_pd(__m256d a, enum intv, enum sgn);
VGETMANTPD __m256d _mm256_mask_getmant_pd(__m256d s, __mmask8 k, __m256d a, enum intv, enum sgn);
VGETMANTPD __m256d _mm256_maskz_getmant_pd(__mmask8 k, __m256d a, enum intv, enum sgn);
VGETMANTPD __m128d _mm_getmant_pd(__m128d a, enum intv, enum sgn);
VGETMANTPD __m128d _mm_mask_getmant_pd(__m128d s, __mmask8 k, __m128d a, enum intv, enum sgn);
VGETMANTPD __m128d _mm_maskz_getmant_pd(__mmask8 k, __m128d a, enum intv, enum sgn);

SIMD Floating-Point Exceptions

Denormal, Invalid

Other Exceptions

See Table 2-46, “Type E2 Class Exception Conditions”; additionally:

#UD	If EVEX.vvvv != 1111B.

VGETMANTPS—Extract Float32 Vector of Normalized Mantissas from Float32 Vector

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.128.66.0F3A.W0 26 /r ib VGETMANTPS xmm1 {k1}{z}, xmm2/m128/m32bcst, imm8	A	V/V	AVX512VL AVX512F	Get normalized mantissa from float32 vector xmm2/m128/m32bcst and store the result in xmm1, using imm8 for sign control and mantissa interval normalization, under writemask.
EVEX.256.66.0F3A.W0 26 /r ib VGETMANTPS ymm1 {k1}{z}, ymm2/m256/m32bcst, imm8	A	V/V	AVX512VL AVX512F	Get normalized mantissa from float32 vector ymm2/m256/m32bcst and store the result in ymm1, using imm8 for sign control and mantissa interval normalization, under writemask.
EVEX.512.66.0F3A.W0 26 /r ib VGETMANTPS zmm1 {k1}{z}, zmm2/m512/m32bcst{sae}, imm8	A	V/V	AVX512F	Get normalized mantissa from float32 vector zmm2/m512/m32bcst and store the result in zmm1, using imm8 for sign control and mantissa interval normalization, under writemask.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	ModRM:r/m (r)	Imm8	NA

Description

Convert single-precision floating values in the source operand (the second operand) to SP FP values with the mantissa normalization and sign control specified by the imm8 byte, see Figure 5-15. The converted results are written to the destination operand (the first operand) using writemask k1. The normalized mantissa is specified by interv (imm8[1:0]) and the sign control (sc) is specified by bits 3:2 of the immediate byte.

The destination operand is a ZMM/YMM/XMM register updated under the writemask. The source operand can be 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.

For each input SP FP value x, The conversion operation is:

$$GetMant(x) = \pm 2^{k}|x.significand|$$

where:

$$1 <= |x.significand| < 2$$

Unbiased exponent k can be either 0 or -1, depending on the interval range defined by interv, the range of the significand and whether the exponent of the source is even or odd. The sign of the final result is determined by sc and the source sign. The encoded value of imm8[1:0] and sign control are shown in Figure 5-15.

Each converted SP FP result is encoded according to the sign control, the unbiased exponent k (adding bias) and a mantissa normalized to the range specified by interv.

The GetMant() function follows Table 5-7 when dealing with floating-point special numbers.

This instruction is writemasked, so only those elements with the corresponding bit set in vector mask register k1 are computed and stored into the destination. Elements in zmm1 with the corresponding bit clear in k1 retain their previous values.

Note: EVEX.vvvv is reserved and must be 1111b, VEX.L must be 0; otherwise instructions will #UD.

Operation

```
def getmant_fp32(src, sign_control, normalization_interval):
    bias := 127
    dst.sign := sign_control[0] ? 0 : src.sign
    signed_one := sign_control[0] ? +1.0 : -1.0
    dst.exp := src.exp
    dst.fraction := src.fraction
    zero := (dst.exp = 0) and ((dst.fraction = 0) or (MXCSR.DAZ=1))
    denormal := (dst.exp = 0) and (dst.fraction != 0) and (MXCSR.DAZ=0)
    infinity := (dst.exp = 0xFF) and (dst.fraction = 0)
    nan := (dst.exp = 0xFF) and (dst.fraction != 0)
    src_signaling := src.fraction[22]
    snan := nan and (src_signaling = 0)
    positive := (src.sign = 0)
    negative := (src.sign = 1)
    if nan:
        if snan:
            MXCSR.IE := 1
        return qnan(src)

    if positive and (zero or infinity):
        return 1.0
    if negative:
        if zero:
            return signed_one
        if infinity:
            if sign_control[1]:
                MXCSR.IE := 1
                return QNaN_Indefinite
            return signed_one
        if sign_control[1]:
            MXCSR.IE := 1
            return QNaN_Indefinite

    if denormal:
        jbit := 0
        dst.exp := bias
        while jbit = 0:
            jbit := dst.fraction[22]
            dst.fraction := dst.fraction << 1
            dst.exp : = dst.exp - 1
        MXCSR.DE := 1

    unbiased_exp := dst.exp - bias
    odd_exp  := unbiased_exp[0]
    signaling_bit := dst.fraction[22]
    if normalization_interval = 0b00:
        dst.exp := bias
    else if normalization_interval = 0b01:
        dst.exp := odd_exp ? bias-1 : bias
    else if normalization_interval = 0b10:
        dst.exp := bias-1
    else if normalization_interval = 0b11:
        dst.exp := signaling_bit ? bias-1 : bias
```

```
    return dst
```

VGETMANTPS (EVEX encoded versions)

```
VGETMANTPS dest{k1}, src, imm8
VL = 128, 256, or 512
KL := VL / 32
sign_control := imm8[3:2]
normalization_interval := imm8[1:0]

FOR i := 0 to KL-1:
    IF k1[i] or *no writemask*:
        IF SRC is memory and (EVEX.b = 1):
            tsrc := src.float[0]
        ELSE:
            tsrc := src.float[i]
        DEST.float[i] := getmant_fp32(tsrc, sign_control, normalization_interval)
    ELSE IF *zeroing*:
        DEST.float[i] := 0
    //else DEST.float[i] remains unchanged

DEST[MAX_VL-1:VL] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

```
VGETMANTPS __m512 _mm512_getmant_ps( __m512 a, enum intv, enum sgn);
VGETMANTPS __m512 _mm512_mask_getmant_ps(__m512 s, __mmask16 k, __m512 a, enum intv, enum sgn;
VGETMANTPS __m512 _mm512_maskz_getmant_ps(__mmask16 k, __m512 a, enum intv, enum sgn);
VGETMANTPS __m512 _mm512_getmant_round_ps( __m512 a, enum intv, enum sgn, int r);
VGETMANTPS __m512 _mm512_mask_getmant_round_ps(__m512 s, __mmask16 k, __m512 a, enum intv, enum sgn, int r);
VGETMANTPS __m512 _mm512_maskz_getmant_round_ps(__mmask16 k, __m512 a, enum intv, enum sgn, int r);
VGETMANTPS __m256 _mm256_getmant_ps( __m256 a, enum intv, enum sgn);
VGETMANTPS __m256 _mm256_mask_getmant_ps(__m256 s, __mmask8 k, __m256 a, enum intv, enum sgn);
VGETMANTPS __m256 _mm256_maskz_getmant_ps( __mmask8 k, __m256 a, enum intv, enum sgn);
VGETMANTPS __m128 _mm_getmant_ps( __m128 a, enum intv, enum sgn);
VGETMANTPS __m128 _mm_mask_getmant_ps(__m128 s, __mmask8 k, __m128 a, enum intv, enum sgn);
VGETMANTPS __m128 _mm_maskz_getmant_ps( __mmask8 k, __m128 a, enum intv, enum sgn);
```

SIMD Floating-Point Exceptions

Denormal, Invalid

Other Exceptions

See Table 2-46, "Type E2 Class Exception Conditions"; additionally:

#UD	If EVEX.vvvv != 1111B.

VGETMANTSD—Extract Float64 of Normalized Mantissas from Float64 Scalar

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.LLIG.66.0F3A.W1 27 /r ib VGETMANTSD xmm1 {k1}{z}, xmm2, xmm3/m64{sae}, imm8	A	V/V	AVX512F	Extract the normalized mantissa of the low float64 element in xmm3/m64 using *imm*8 for sign control and mantissa interval normalization. Store the mantissa to xmm1 under the writemask k1 and merge with the other elements of xmm2.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

Description

Convert the double-precision floating values in the low quadword element of the second source operand (the third operand) to DP FP value with the mantissa normalization and sign control specified by the imm8 byte, see Figure 5-15. The converted result is written to the low quadword element of the destination operand (the first operand) using writemask k1. Bits (127:64) of the XMM register destination are copied from corresponding bits in the first source operand. The normalized mantissa is specified by interv (imm8[1:0]) and the sign control (sc) is specified by bits 3:2 of the immediate byte.

The conversion operation is:

$$GetMant(x) = \pm 2^{k}|x.significand|$$

where:

$$1 <= |x.significand| < 2$$

Unbiased exponent k can be either 0 or -1, depending on the interval range defined by interv, the range of the significand and whether the exponent of the source is even or odd. The sign of the final result is determined by sc and the source sign. The encoded value of imm8[1:0] and sign control are shown in Figure 5-15.

The converted DP FP result is encoded according to the sign control, the unbiased exponent k (adding bias) and a mantissa normalized to the range specified by interv.

The GetMant() function follows Table 5-7 when dealing with floating-point special numbers.

If writemasking is used, the low quadword element of the destination operand is conditionally updated depending on the value of writemask register k1. If writemasking is not used, the low quadword element of the destination operand is unconditionally updated.

Operation

```
// getmant_fp64(src, sign_control, normalization_interval) is defined in the operation section of VGETMANTPD

VGETMANTSD (EVEX encoded version)
SignCtrl[1:0] := IMM8[3:2];
Interv[1:0] := IMM8[1:0];
IF k1[0] OR *no writemask*
    THEN DEST[63:0] :=
            getmant_fp64(src, sign_control, normalization_interval)
    ELSE
        IF *merging-masking*                    ; merging-masking
            THEN *DEST[63:0] remains unchanged*
            ELSE                                ; zeroing-masking
                DEST[63:0] := 0
        FI
FI;
DEST[127:64] := SRC1[127:64]
DEST[MAXVL-1:128] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

```
VGETMANTSD __m128d _mm_getmant_sd( __m128d a, __m128 b, enum intv, enum sgn);
VGETMANTSD __m128d _mm_mask_getmant_sd(__m128d s, __mmask8 k, __m128d a, __m128d b, enum intv, enum sgn);
VGETMANTSD __m128d _mm_maskz_getmant_sd( __mmask8 k, __m128 a, __m128d b, enum intv, enum sgn);
VGETMANTSD __m128d _mm_getmant_round_sd( __m128d a, __m128 b, enum intv, enum sgn, int r);
VGETMANTSD __m128d _mm_mask_getmant_round_sd(__m128d s, __mmask8 k, __m128d a, __m128d b, enum intv, enum sgn, int r);
VGETMANTSD __m128d _mm_maskz_getmant_round_sd( __mmask8 k, __m128d a, __m128d b, enum intv, enum sgn, int r);
```

SIMD Floating-Point Exceptions

Denormal, Invalid

Other Exceptions

See Table 2-47, "Type E3 Class Exception Conditions".

VGETMANTSS—Extract Float32 Vector of Normalized Mantissa from Float32 Vector

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.LLIG.66.0F3A.W0 27 /r ib VGETMANTSS xmm1 {k1}{z}, xmm2, xmm3/m32{sae}, imm8	A	V/V	AVX512F	Extract the normalized mantissa from the low float32 element of xmm3/m32 using imm8 for sign control and mantissa interval normalization, store the mantissa to xmm1 under the writemask k1 and merge with the other elements of xmm2.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	Tuple1 Scalar	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

Description

Convert the single-precision floating values in the low doubleword element of the second source operand (the third operand) to SP FP value with the mantissa normalization and sign control specified by the imm8 byte, see Figure 5-15. The converted result is written to the low doubleword element of the destination operand (the first operand) using writemask k1. Bits (127:32) of the XMM register destination are copied from corresponding bits in the first source operand. The normalized mantissa is specified by interv (imm8[1:0]) and the sign control (sc) is specified by bits 3:2 of the immediate byte.

The conversion operation is:

$$GetMant(x) = \pm 2^{k}|x.significand|$$

where:

$$1 <= |x.significand| < 2$$

Unbiased exponent k can be either 0 or -1, depending on the interval range defined by interv, the range of the significand and whether the exponent of the source is even or odd. The sign of the final result is determined by sc and the source sign. The encoded value of imm8[1:0] and sign control are shown in Figure 5-15.

The converted SP FP result is encoded according to the sign control, the unbiased exponent k (adding bias) and a mantissa normalized to the range specified by interv.

The GetMant() function follows Table 5-7 when dealing with floating-point special numbers.

If writemasking is used, the low doubleword element of the destination operand is conditionally updated depending on the value of writemask register k1. If writemasking is not used, the low doubleword element of the destination operand is unconditionally updated.

Operation

```
// getmant_fp32(src, sign_control, normalization_interval) is defined in the operation section of VGETMANTPS

VGETMANTSS (EVEX encoded version)
SignCtrl[1:0] := IMM8[3:2];
Interv[1:0] := IMM8[1:0];
IF k1[0] OR *no writemask*
    THEN DEST[31:0] :=
            getmant_fp32(src, sign_control, normalization_interval)
    ELSE
        IF *merging-masking*                    ; merging-masking
            THEN *DEST[31:0] remains unchanged*
            ELSE                                ; zeroing-masking
                DEST[31:0] := 0
        FI
FI;
DEST[127:32] := SRC1[127:32]
DEST[MAXVL-1:128] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

```
VGETMANTSS __m128 _mm_getmant_ss( __m128 a, __m128 b, enum intv, enum sgn);
VGETMANTSS __m128 _mm_mask_getmant_ss(__m128 s, __mmask8 k, __m128 a, __m128 b, enum intv, enum sgn);
VGETMANTSS __m128 _mm_maskz_getmant_ss( __mmask8 k, __m128 a, __m128 b, enum intv, enum sgn);
VGETMANTSS __m128 _mm_getmant_round_ss( __m128 a, __m128 b, enum intv, enum sgn, int r);
VGETMANTSS __m128 _mm_mask_getmant_round_ss(__m128 s, __mmask8 k, __m128 a, __m128 b, enum intv, enum sgn, int r);
VGETMANTSS __m128 _mm_maskz_getmant_round_ss( __mmask8 k, __m128 a, __m128 b, enum intv, enum sgn, int r);
```

SIMD Floating-Point Exceptions

Denormal, Invalid

Other Exceptions

See Table 2-47, "Type E3 Class Exception Conditions".

VPDPBUSD — Multiply and Add Unsigned and Signed Bytes

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F38.W0 50 /r VPDPBUSD xmm1, xmm2, xmm3/m128	A	V/V	AVX-VNNI	Multiply groups of 4 pairs of signed bytes in xmm3/m128 with corresponding unsigned bytes of xmm2, summing those products and adding them to doubleword result in xmm1.
VEX.256.66.0F38.W0 50 /r VPDPBUSD ymm1, ymm2, ymm3/m256	A	V/V	AVX-VNNI	Multiply groups of 4 pairs of signed bytes in ymm3/m256 with corresponding unsigned bytes of ymm2, summing those products and adding them to doubleword result in ymm1.
EVEX.128.66.0F38.W0 50 /r VPDPBUSD xmm1{k1}{z}, xmm2, xmm3/m128/m32bcst	B	V/V	AVX512_VNNI AVX512VL	Multiply groups of 4 pairs of signed bytes in xmm3/m128/m32bcst with corresponding unsigned bytes of xmm2, summing those products and adding them to doubleword result in xmm1 under writemask k1.
EVEX.256.66.0F38.W0 50 /r VPDPBUSD ymm1{k1}{z}, ymm2, ymm3/m256/m32bcst	B	V/V	AVX512_VNNI AVX512VL	Multiply groups of 4 pairs of signed bytes in ymm3/m256/m32bcst with corresponding unsigned bytes of ymm2, summing those products and adding them to doubleword result in ymm1 under writemask k1.
EVEX.512.66.0F38.W0 50 /r VPDPBUSD zmm1{k1}{z}, zmm2, zmm3/m512/m32bcst	B	V/V	AVX512_VNNI	Multiply groups of 4 pairs of signed bytes in zmm3/m512/m32bcst with corresponding unsigned bytes of zmm2, summing those products and adding them to doubleword result in zmm1 under writemask k1.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	ModRM:reg (r, w)	VEX.vvvv (r)	ModRM:r/m (r)	NA
B	Full	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

Description

Multiplies the individual unsigned bytes of the first source operand by the corresponding signed bytes of the second source operand, producing intermediate signed word results. The word results are then summed and accumulated in the destination dword element size operand.

This instruction supports memory fault suppression.

Operation

VPDPBUSD dest, src1, src2 (VEX encoded versions)

```
VL=(128, 256)
KL=VL/32

ORIGDEST := DEST
FOR i := 0 TO KL-1:

    // Extending to 16b
    // src1extend := ZERO_EXTEND
    // src2extend := SIGN_EXTEND

    p1word := src1extend(SRC1.byte[4*i+0]) * src2extend(SRC2.byte[4*i+0])
    p2word := src1extend(SRC1.byte[4*i+1]) * src2extend(SRC2.byte[4*i+1])
    p3word := src1extend(SRC1.byte[4*i+2]) * src2extend(SRC2.byte[4*i+2])
    p4word := src1extend(SRC1.byte[4*i+3]) * src2extend(SRC2.byte[4*i+3])
    DEST.dword[i] := ORIGDEST.dword[i] + p1word + p2word + p3word + p4word

DEST[MAX_VL-1:VL] := 0
```

VPDPBUSD dest, src1, src2 (EVEX encoded versions)

```
(KL,VL)=(4,128), (8,256), (16,512)
ORIGDEST := DEST
FOR i := 0 TO KL-1:
    IF k1[i] or *no writemask*:
        // Byte elements of SRC1 are zero-extended to 16b and
        // byte elements of SRC2 are sign extended to 16b before multiplication.
        IF SRC2 is memory and EVEX.b == 1:
            t := SRC2.dword[0]
        ELSE:
            t := SRC2.dword[i]
        p1word := ZERO_EXTEND(SRC1.byte[4*i]) * SIGN_EXTEND(t.byte[0])
        p2word := ZERO_EXTEND(SRC1.byte[4*i+1]) * SIGN_EXTEND(t.byte[1])
        p3word := ZERO_EXTEND(SRC1.byte[4*i+2]) * SIGN_EXTEND(t.byte[2])
        p4word := ZERO_EXTEND(SRC1.byte[4*i+3]) * SIGN_EXTEND(t.byte[3])
        DEST.dword[i] := ORIGDEST.dword[i] + p1word + p2word + p3word + p4word
    ELSE IF *zeroing*:
        DEST.dword[i] := 0
    ELSE:    // Merge masking, dest element unchanged
        DEST.dword[i] := ORIGDEST.dword[i]
DEST[MAX_VL-1:VL] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

```
VPDPBUSD __m128i _mm_dpbusd_avx_epi32(__m128i, __m128i, __m128i);
VPDPBUSD __m128i _mm_dpbusd_epi32(__m128i, __m128i, __m128i);
VPDPBUSD __m128i _mm_mask_dpbusd_epi32(__m128i, __mmask8, __m128i, __m128i);
VPDPBUSD __m128i _mm_maskz_dpbusd_epi32(__mmask8, __m128i, __m128i, __m128i);
VPDPBUSD __m256i _mm256_dpbusd_avx_epi32(__m256i, __m256i, __m256i);
VPDPBUSD __m256i _mm256_dpbusd_epi32(__m256i, __m256i, __m256i);
VPDPBUSD __m256i _mm256_mask_dpbusd_epi32(__m256i, __mmask8, __m256i, __m256i);
VPDPBUSD __m256i _mm256_maskz_dpbusd_epi32(__mmask8, __m256i, __m256i, __m256i);
VPDPBUSD __m512i _mm512_dpbusd_epi32(__m512i, __m512i, __m512i);
VPDPBUSD __m512i _mm512_mask_dpbusd_epi32(__m512i, __mmask16, __m512i, __m512i);
VPDPBUSD __m512i _mm512_maskz_dpbusd_epi32(__mmask16, __m512i, __m512i, __m512i);
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

Non-EVEX-encoded instruction, see Table 2-21, "Type 4 Class Exception Conditions".

EVEX-encoded instruction, see Table 2-49, "Type E4 Class Exception Conditions".

VPDPBUSDS — Multiply and Add Unsigned and Signed Bytes with Saturation

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F38.W0 51 /r VPDPBUSDS xmm1, xmm2, xmm3/m128	A	V/V	AVX-VNNI	Multiply groups of 4 pairs signed bytes in xmm3/m128 with corresponding unsigned bytes of xmm2, summing those products and adding them to doubleword result, with signed saturation in xmm1.
VEX.256.66.0F38.W0 51 /r VPDPBUSDS ymm1, ymm2, ymm3/m256	A	V/V	AVX-VNNI	Multiply groups of 4 pairs signed bytes in ymm3/m256 with corresponding unsigned bytes of ymm2, summing those products and adding them to doubleword result, with signed saturation in ymm1.
EVEX.128.66.0F38.W0 51 /r VPDPBUSDS xmm1{k1}{z}, xmm2, xmm3/m128/m32bcst	B	V/V	AVX512_VNNI AVX512VL	Multiply groups of 4 pairs signed bytes in xmm3/m128/m32bcst with corresponding unsigned bytes of xmm2, summing those products and adding them to doubleword result, with signed saturation in xmm1, under writemask k1.
EVEX.256.66.0F38.W0 51 /r VPDPBUSDS ymm1{k1}{z}, ymm2, ymm3/m256/m32bcst	B	V/V	AVX512_VNNI AVX512VL	Multiply groups of 4 pairs signed bytes in ymm3/m256/m32bcst with corresponding unsigned bytes of ymm2, summing those products and adding them to doubleword result, with signed saturation in ymm1, under writemask k1.
EVEX.512.66.0F38.W0 51 /r VPDPBUSDS zmm1{k1}{z}, zmm2, zmm3/m512/m32bcst	B	V/V	AVX512_VNNI	Multiply groups of 4 pairs signed bytes in zmm3/m512/m32bcst with corresponding unsigned bytes of zmm2, summing those products and adding them to doubleword result, with signed saturation in zmm1, under writemask k1.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	ModRM:reg (r, w)	VEX.vvvv (r)	ModRM:r/m (r)	NA
B	Full	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

Description

Multiplies the individual unsigned bytes of the first source operand by the corresponding signed bytes of the second source operand, producing intermediate signed word results. The word results are then summed and accumulated in the destination dword element size operand. If the intermediate sum overflows a 32b signed number the result is saturated to either 0x7FFF_FFFF for positive numbers of 0x8000_0000 for negative numbers.

This instruction supports memory fault suppression.

Operation

VPDPBUSDS dest, src1, src2 (VEX encoded versions)

```
VL=(128, 256)
KL=VL/32

ORIGDEST := DEST
FOR i := 0 TO KL-1:

    // Extending to 16b
    // src1extend := ZERO_EXTEND
    // src2extend := SIGN_EXTEND

    p1word := src1extend(SRC1.byte[4*i+0]) * src2extend(SRC2.byte[4*i+0])
    p2word := src1extend(SRC1.byte[4*i+1]) * src2extend(SRC2.byte[4*i+1])
    p3word := src1extend(SRC1.byte[4*i+2]) * src2extend(SRC2.byte[4*i+2])
    p4word := src1extend(SRC1.byte[4*i+3]) * src2extend(SRC2.byte[4*i+3])
    DEST.dword[i] := SIGNED_DWORD_SATURATE(ORIGDEST.dword[i] + p1word + p2word + p3word + p4word)

DEST[MAX_VL-1:VL] := 0
```

VPDPBUSDS dest, src1, src2 (EVEX encoded versions)

```
(KL,VL)=(4,128), (8,256), (16,512)
ORIGDEST := DEST
FOR i := 0 TO KL-1:
    IF k1[i] or *no writemask*:
        // Byte elements of SRC1 are zero-extended to 16b and
        // byte elements of SRC2 are sign extended to 16b before multiplication.
        IF SRC2 is memory and EVEX.b == 1:
            t := SRC2.dword[0]
        ELSE:
            t := SRC2.dword[i]
        p1word := ZERO_EXTEND(SRC1.byte[4*i]) * SIGN_EXTEND(t.byte[0])
        p2word := ZERO_EXTEND(SRC1.byte[4*i+1]) * SIGN_EXTEND(t.byte[1])
        p3word := ZERO_EXTEND(SRC1.byte[4*i+2]) * SIGN_EXTEND(t.byte[2])
        p4word := ZERO_EXTEND(SRC1.byte[4*i+3]) *SIGN_EXTEND(t.byte[3])
        DEST.dword[i] := SIGNED_DWORD_SATURATE(ORIGDEST.dword[i] + p1word + p2word + p3word + p4word)
    ELSE IF *zeroing*:
        DEST.dword[i] := 0
    ELSE:    // Merge masking, dest element unchanged
        DEST.dword[i] := ORIGDEST.dword[i]
DEST[MAX_VL-1:VL] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

```
VPDPBUSDS __m128i _mm_dpbusds_avx_epi32(__m128i, __m128i, __m128i);
VPDPBUSDS __m128i _mm_dpbusds_epi32(__m128i, __m128i, __m128i);
VPDPBUSDS __m128i _mm_mask_dpbusds_epi32(__m128i, __mmask8, __m128i, __m128i);
VPDPBUSDS __m128i _mm_maskz_dpbusds_epi32(__mmask8, __m128i, __m128i, __m128i);
VPDPBUSDS __m256i _mm256_dpbusds_avx_epi32(__m256i, __m256i, __m256i);
VPDPBUSDS __m256i _mm256_dpbusds_epi32(__m256i, __m256i, __m256i);
VPDPBUSDS __m256i _mm256_mask_dpbusds_epi32(__m256i, __mmask8, __m256i, __m256i);
VPDPBUSDS __m256i _mm256_maskz_dpbusds_epi32(__mmask8, __m256i, __m256i, __m256i);
VPDPBUSDS __m512i _mm512_dpbusds_epi32(__m512i, __m512i, __m512i);
VPDPBUSDS __m512i _mm512_mask_dpbusds_epi32(__m512i, __mmask16, __m512i, __m512i);
VPDPBUSDS __m512i _mm512_maskz_dpbusds_epi32(__mmask16, __m512i, __m512i, __m512i);
```

SIMD Floating-Point Exceptions

None.

Other Exceptions

Non-EVEX-encoded instruction, see Table 2-21, "Type 4 Class Exception Conditions".

EVEX-encoded instruction, see Table 2-49, "Type E4 Class Exception Conditions".

VPDPWSSD — Multiply and Add Signed Word Integers

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F38.W0 52 /r VPDPWSSD xmm1, xmm2, xmm3/m128	A	V/V	AVX-VNNI	Multiply groups of 2 pairs signed words in xmm3/m128 with corresponding signed words of xmm2, summing those products and adding them to doubleword result in xmm1.
VEX.256.66.0F38.W0 52 /r VPDPWSSD ymm1, ymm2, ymm3/m256	A	V/V	AVX-VNNI	Multiply groups of 2 pairs signed words in ymm3/m256 with corresponding signed words of ymm2, summing those products and adding them to doubleword result in ymm1.
EVEX.128.66.0F38.W0 52 /r VPDPWSSD xmm1{k1}{z}, xmm2, xmm3/m128/m32bcst	B	V/V	AVX512_VNNI AVX512VL	Multiply groups of 2 pairs signed words in xmm3/m128/m32bcst with corresponding signed words of xmm2, summing those products and adding them to doubleword result in xmm1, under writemask k1.
EVEX.256.66.0F38.W0 52 /r VPDPWSSD ymm1{k1}{z}, ymm2, ymm3/m256/m32bcst	B	V/V	AVX512_VNNI AVX512VL	Multiply groups of 2 pairs signed words in ymm3/m256/m32bcst with corresponding signed words of ymm2, summing those products and adding them to doubleword result in ymm1, under writemask k1.
EVEX.512.66.0F38.W0 52 /r VPDPWSSD zmm1{k1}{z}, zmm2, zmm3/m512/m32bcst	B	V/V	AVX512_VNNI	Multiply groups of 2 pairs signed words in zmm3/m512/m32bcst with corresponding signed words of zmm2, summing those products and adding them to doubleword result in zmm1, under writemask k1.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	ModRM:reg (r, w)	VEX.vvvv (r)	ModRM:r/m (r)	NA
B	Full	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

Description

Multiplies the individual signed words of the first source operand by the corresponding signed words of the second source operand, producing intermediate signed, doubleword results. The adjacent doubleword results are then summed and accumulated in the destination operand.

This instruction supports memory fault suppression.

Operation

VPDPWSSD dest, src1, src2 (VEX encoded versions)

```
VL=(128, 256)
KL=VL/32
ORIGDEST := DEST
FOR i := 0 TO KL-1:
    p1dword := SIGN_EXTEND(SRC1.word[2*i+0]) * SIGN_EXTEND(SRC2.word[2*i+0] )
    p2dword := SIGN_EXTEND(SRC1.word[2*i+1]) * SIGN_EXTEND(SRC2.word[2*i+1] )
    DEST.dword[i] := ORIGDEST.dword[i] + p1dword + p2dword
DEST[MAX_VL-1:VL] := 0
```

VPDPWSSD dest, src1, src2 (EVEX encoded versions)

```
(KL,VL)=(4,128), (8,256), (16,512)
ORIGDEST := DEST
FOR i := 0 TO KL-1:
    IF k1[i] or *no writemask*:
        IF SRC2 is memory and EVEX.b == 1:
            t := SRC2.dword[0]
        ELSE:
            t := SRC2.dword[i]
        p1dword := SIGN_EXTEND(SRC1.word[2*i]) * SIGN_EXTEND(t.word[0])
        p2dword := SIGN_EXTEND(SRC1.word[2*i+1]) * SIGN_EXTEND(t.word[1])
        DEST.dword[i] := ORIGDEST.dword[i] + p1dword + p2dword
    ELSE IF *zeroing*:
        DEST.dword[i] := 0
    ELSE:       // Merge masking, dest element unchanged
        DEST.dword[i] := ORIGDEST.dword[i]
DEST[MAX_VL-1:VL] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

VPDPWSSD __m128i _mm_dpwssd_avx_epi32(__m128i, __m128i, __m128i);
VPDPWSSD __m128i _mm_dpwssd_epi32(__m128i, __m128i, __m128i);
VPDPWSSD __m128i _mm_mask_dpwssd_epi32(__m128i, __mmask8, __m128i, __m128i);
VPDPWSSD __m128i _mm_maskz_dpwssd_epi32(__mmask8, __m128i, __m128i, __m128i);
VPDPWSSD __m256i _mm256_dpwssd_avx_epi32(__m256i, __m256i, __m256i);
VPDPWSSD __m256i _mm256_dpwssd_epi32(__m256i, __m256i, __m256i);
VPDPWSSD __m256i _mm256_mask_dpwssd_epi32(__m256i, __mmask8, __m256i, __m256i);
VPDPWSSD __m256i _mm256_maskz_dpwssd_epi32(__mmask8, __m256i, __m256i, __m256i);
VPDPWSSD __m512i _mm512_dpwssd_epi32(__m512i, __m512i, __m512i);
VPDPWSSD __m512i _mm512_mask_dpwssd_epi32(__m512i, __mmask16, __m512i, __m512i);
VPDPWSSD __m512i _mm512_maskz_dpwssd_epi32(__mmask16, __m512i, __m512i, __m512i);

SIMD Floating-Point Exceptions

None.

Other Exceptions

Non-EVEX-encoded instruction, see Table 2-21, "Type 4 Class Exception Conditions".

EVEX-encoded instruction, see Table 2-49, "Type E4 Class Exception Conditions".

VPDPWSSDS — Multiply and Add Signed Word Integers with Saturation

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F38.W0 53 /r VPDPWSSDS xmm1, xmm2, xmm3/m128	A	V/V	AVX-VNNI	Multiply groups of 2 pairs of signed words in xmm3/m128 with corresponding signed words of xmm2, summing those products and adding them to doubleword result in xmm1, with signed saturation.
VEX.256.66.0F38.W0 53 /r VPDPWSSDS ymm1, ymm2, ymm3/m256	A	V/V	AVX-VNNI	Multiply groups of 2 pairs of signed words in ymm3/m256 with corresponding signed words of ymm2, summing those products and adding them to doubleword result in ymm1, with signed saturation.
EVEX.128.66.0F38.W0 53 /r VPDPWSSDS xmm1{k1}{z}, xmm2, xmm3/m128/m32bcst	B	V/V	AVX512_VNNI AVX512VL	Multiply groups of 2 pairs of signed words in xmm3/m128/m32bcst with corresponding signed words of xmm2, summing those products and adding them to doubleword result in xmm1, with signed saturation, under writemask k1.
EVEX.256.66.0F38.W0 53 /r VPDPWSSDS ymm1{k1}{z}, ymm2, ymm3/m256/m32bcst	B	V/V	AVX512_VNNI AVX512VL	Multiply groups of 2 pairs of signed words in ymm3/m256/m32bcst with corresponding signed words of ymm2, summing those products and adding them to doubleword result in ymm1, with signed saturation, under writemask k1.
EVEX.512.66.0F38.W0 53 /r VPDPWSSDS zmm1{k1}{z}, zmm2, zmm3/m512/m32bcst	B	V/V	AVX512_VNNI	Multiply groups of 2 pairs of signed words in zmm3/m512/m32bcst with corresponding signed words of zmm2, summing those products and adding them to doubleword result in zmm1, with signed saturation, under writemask k1.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	ModRM:reg (r, w)	VEX.vvvv (r)	ModRM:r/m (r)	NA
B	Full	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

Description

Multiplies the individual signed words of the first source operand by the corresponding signed words of the second source operand, producing intermediate signed, doubleword results. The adjacent doubleword results are then summed and accumulated in the destination operand. If the intermediate sum overflows a 32b signed number, the result is saturated to either 0x7FFF_FFFF for positive numbers of 0x8000_0000 for negative numbers.

This instruction supports memory fault suppression.

Operation

VPDPWSSDS dest, src1, src2 (VEX encoded versions)

```
VL=(128, 256)
KL=VL/32
ORIGDEST := DEST
FOR i := 0 TO KL-1:
    p1dword := SIGN_EXTEND(SRC1.word[2*i+0]) * SIGN_EXTEND(SRC2.word[2*i+0])
    p2dword := SIGN_EXTEND(SRC1.word[2*i+1]) * SIGN_EXTEND(SRC2.word[2*i+1])
    DEST.dword[i] := SIGNED_DWORD_SATURATE(ORIGDEST.dword[i] + p1dword + p2dword)
DEST[MAX_VL-1:VL] := 0
```

VPDPWSSDS dest, src1, src2 (EVEX encoded versions)

```
(KL,VL)=(4,128), (8,256), (16,512)
ORIGDEST := DEST
FOR i := 0 TO KL-1:
    IF k1[i] or *no writemask*:
        IF SRC2 is memory and EVEX.b == 1:
            t := SRC2.dword[0]
        ELSE:
            t := SRC2.dword[i]
        p1dword := SIGN_EXTEND(SRC1.word[2*i]) * SIGN_EXTEND(t.word[0])
        p2dword := SIGN_EXTEND(SRC1.word[2*i+1]) * SIGN_EXTEND(t.word[1])
        DEST.dword[i] := SIGNED_DWORD_SATURATE(ORIGDEST.dword[i] + p1dword + p2dword)
    ELSE IF *zeroing*:
        DEST.dword[i] := 0
    ELSE:     // Merge masking, dest element unchanged
        DEST.dword[i] := ORIGDEST.dword[i]
DEST[MAX_VL-1:VL] := 0
```

Intel C/C++ Compiler Intrinsic Equivalent

VPDPWSSDS __m128i _mm_dpwssds_avx_epi32(__m128i, __m128i, __m128i);
VPDPWSSDS __m128i _mm_dpwssds_epi32(__m128i, __m128i, __m128i);
VPDPWSSDS __m128i _mm_mask_dpwssd_epi32(__m128i, __mmask8, __m128i, __m128i);
VPDPWSSDS __m128i _mm_maskz_dpwssd_epi32(__mmask8, __m128i, __m128i, __m128i);
VPDPWSSDS __m256i _mm256_dpwssds_avx_epi32(__m256i, __m256i, __m256i);
VPDPWSSDS __m256i _mm256_dpwssd_epi32(__m256i, __m256i, __m256i);
VPDPWSSDS __m256i _mm256_mask_dpwssd_epi32(__m256i, __mmask8, __m256i, __m256i);
VPDPWSSDS __m256i _mm256_maskz_dpwssd_epi32(__mmask8, __m256i, __m256i, __m256i);
VPDPWSSDS __m512i _mm512_dpwssd_epi32(__m512i, __m512i, __m512i);
VPDPWSSDS __m512i _mm512_mask_dpwssd_epi32(__m512i, __mmask16, __m512i, __m512i);
VPDPWSSDS __m512i _mm512_maskz_dpwssd_epi32(__mmask16, __m512i, __m512i, __m512i);

SIMD Floating-Point Exceptions

None.

Other Exceptions

Non-EVEX-encoded instruction, see Table 2-21, "Type 4 Class Exception Conditions".

EVEX-encoded instruction, see Table 2-49, "Type E4 Class Exception Conditions".

WRSSD/WRSSQ—Write to Shadow Stack

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 38 F6 !(11):rrr:bbb WRSSD m32, r32	MR	V/V	CET_SS	Write 4 bytes to shadow stack.
REX.W 0F 38 F6 !(11):rrr:bbb WRSSQ m64, r64	MR	V/N.E.	CET_SS	Write 8 bytes to shadow stack.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MR	ModRM:r/m (w)	ModRM:reg (r)	NA	NA

Description

Writes bytes in register source to the shadow stack.

Operation

```
IF CPL = 3
    IF (CR4.CET & IA32_U_CET.SH_STK_EN) = 0
        THEN #UD; FI;
    IF (IA32_U_CET.WR_SHSTK_EN) = 0
        THEN #UD; FI;
ELSE
    IF (CR4.CET & IA32_S_CET.SH_STK_EN) = 0
        THEN #UD; FI;
    IF (IA32_S_CET.WR_SHSTK_EN) = 0
        THEN #UD; FI;
FI;
DEST_LA = Linear_Address(mem operand)
IF (operand size is 64 bit)
    THEN
        (* Destination not 8B aligned *)
        IF DEST_LA[2:0]
            THEN GP(0); FI;
        Shadow_stack_store 8 bytes of SRC to DEST_LA;
    ELSE
        (* Destination not 4B aligned *)
        IF DEST_LA[1:0]
            THEN GP(0); FI;
        Shadow_stack_store 4 bytes of SRC[31:0] to DEST_LA;
FI;
```

Flags Affected

None.

C/C++ Compiler Intrinsic Equivalent

WRSSD void _wrssd(__int32, void *);

WRSSQ void _wrssq(__int64, 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.
	If CPL = 3 and IA32_U_CET.WR_SHSTK_EN = 0.
	If CPL < 3 and IA32_S_CET.WR_SHSTK_EN = 0.
#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If destination is located in a non-writeable segment.
	If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.
	If linear address of destination is not 4 byte aligned.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs if destination is not a user shadow stack when CPL3 and not a supervisor shadow stack when CPL < 3.
	Other terminal and non-terminal faults.

Real-Address Mode Exceptions

#UD	The WRSS instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The WRSS instruction is not recognized in virtual-8086 mode.

Compatibility 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.
	If CPL = 3 and IA32_U_CET.WR_SHSTK_EN = 0.
	If CPL < 3 and IA32_S_CET.WR_SHSTK_EN = 0.
#PF(fault-code)	If a page fault occurs if destination is not a user shadow stack when CPL3 and not a supervisor shadow stack when CPL < 3.
	Other terminal and non-terminal faults.

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.
	If CPL = 3 and IA32_U_CET.WR_SHSTK_EN = 0.
	If CPL < 3 and IA32_S_CET.WR_SHSTK_EN = 0.
#GP(0)	If a memory address is in a non-canonical form.
	If linear address of destination is not 4 byte aligned.
#PF(fault-code)	If a page fault occurs if destination is not a user shadow stack when CPL3 and not a supervisor shadow stack when CPL < 3.
	Other terminal and non-terminal faults.

WRUSSD/WRUSSQ—Write to User Shadow Stack

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 38 F5 !(11):rrr:bbb WRUSSD m32, r32	MR	V/V	CET_SS	Write 4 bytes to shadow stack.
66 REX.W 0F 38 F5 !(11):rrr:bbb WRUSSQ m64, r64	MR	V/N.E.	CET_SS	Write 8 bytes to shadow stack.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MR	ModRM:r/m (w)	ModRM:reg (r)	NA	NA

Description

Writes bytes in register source to a user shadow stack page. The WRUSS instruction can be executed only if CPL = 0, however the processor treats its shadow-stack accesses as user accesses.

Operation

```
IF CR4.CET = 0
    THEN #UD; FI;
IF CPL > 0
    THEN #GP(0); FI;
DEST_LA = Linear_Address(mem operand)
IF (operand size is 64 bit)
    THEN
        (* Destination not 8B aligned *)
        IF DEST_LA[2:0]
            THEN GP(0); FI;
        Shadow_stack_store 8 bytes of SRC to DEST_LA as user-mode access;
    ELSE
        (* Destination not 4B aligned *)
        IF DEST_LA[1:0]
            THEN GP(0); FI;
        Shadow_stack_store 4 bytes of SRC[31:0] to DEST_LA as user-mode access;
FI;
```

Flags Affected

None.

C/C++ Compiler Intrinsic Equivalent

WRUSSD void _wrussd(__int32, void *);

WRUSSQ void _wrussq(__int64, void *);

Protected Mode Exceptions

#UD	If the LOCK prefix is used.
	If CR4.CET = 0.
#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If destination is located in a non-writeable segment.
	If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.
	If linear address of destination is not 4 byte aligned.
	If CPL is not 0.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If destination is not a user shadow stack.
	Other terminal and non-terminal faults.

Real-Address Mode Exceptions

#UD	The WRUSS instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The WRUSS instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	If the LOCK prefix is used.
	If CR4.CET = 0.
#GP(0)	If a memory address is in a non-canonical form.
	If linear address of destination is not 4 byte aligned.
	If CPL is not 0.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	If destination is not a user shadow stack.
	Other terminal and non-terminal faults.

64-Bit Mode Exceptions

#UD	If the LOCK prefix is used.
	If CR4.CET = 0.
#GP(0)	If a memory address is in a non-canonical form.
	If linear address of destination is not 4 byte aligned.
	If CPL is not 0.
#PF(fault-code)	If destination is not a user shadow stack.
	Other terminal and non-terminal faults.

10. Updates to Chapter 1, Volume 3A

Change bars and green text show changes to Chapter 1 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

Changes to this chapter: Updated section 1.1, "Intel® 64 and IA-32 Processors Covered in this Manual" to add the 12th generation Intel® Core™ processor. Added new Chapter 18 to Section 1.2, "Overview of the System Programming Guide". Additional minor corrections in chapter.

CHAPTER 1
ABOUT THIS MANUAL

The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1* (order number 253668), the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2* (order number 253669), the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*: *System Programming Guide, Part 3* (order number 326019), and the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D:System Programming Guide, Part 4* (order number 332831) are part of a set that describes the architecture and programming environment of Intel 64 and IA-32 Architecture processors. The other volumes in this set are:

- *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*: *Basic Architecture* (order number 253665).
- *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D: Instruction Set Reference* (order numbers 253666, 253667, 326018 and 334569).
- The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers* (order number 335592).

The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1,* describes the basic architecture and programming environment of Intel 64 and IA-32 processors. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D,* describe the instruction set of the processor and the opcode structure. These volumes apply to application programmers and to programmers who write operating systems or executives. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A, 3B, 3C & 3D*, describe the operating-system support environment of Intel 64 and IA-32 processors. These volumes target operating-system and BIOS designers. In addition, *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*, and *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* address the programming environment for classes of software that host operating systems. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4,* describes the model-specific registers of Intel 64 and IA-32 processors.

1.1 INTEL® 64 AND IA-32 PROCESSORS COVERED IN THIS MANUAL

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™2 Duo processor
- Intel® Core™2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series

- Intel® Core™2 Extreme processor X7000 and X6800 series
- Intel® Core™2 Extreme QX6000 series
- Intel® Xeon® processor 7100 series
- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Core™2 Extreme QX9000 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel Atom® processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are built from 45 nm and 32 nm processes.
- Intel® Core™ i7 processor
- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Core™ i7-3930K processor
- 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series
- Intel® Xeon® processor E3-1200 product family
- Intel® Xeon® processor E5-2400/1400 product family
- Intel® Xeon® processor E5-4600/2600/1600 product family
- 3rd generation Intel® Core™ processors
- Intel® Xeon® processor E3-1200 v2 product family
- Intel® Xeon® processor E5-2400/1400 v2 product families
- Intel® Xeon® processor E5-4600/2600/1600 v2 product families
- Intel® Xeon® processor E7-8800/4800/2800 v2 product families
- 4th generation Intel® Core™ processors
- The Intel® Core™ M processor family
- Intel® Core™ i7-59xx Processor Extreme Edition
- Intel® Core™ i7-49xx Processor Extreme Edition
- Intel® Xeon® processor E3-1200 v3 product family
- Intel® Xeon® processor E5-2600/1600 v3 product families
- 5th generation Intel® Core™ processors
- Intel® Xeon® processor D-1500 product family
- Intel® Xeon® processor E5 v4 family
- Intel Atom® processor X7-Z8000 and X5-Z8000 series
- Intel Atom® processor Z3400 series
- Intel Atom® processor Z3500 series
- 6th generation Intel® Core™ processors
- Intel® Xeon® processor E3-1500m v5 product family
- 7th generation Intel® Core™ processors
- Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series
- Intel® Xeon® Processor Scalable Family
- 8th generation Intel® Core™ processors
- Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series

- Intel® Xeon® E processors
- 9th generation Intel® Core™ processors
- 2nd generation Intel® Xeon® Processor Scalable Family
- 10th generation Intel® Core™ processors
- 11th generation Intel® Core™ processors
- 3rd generation Intel® Xeon® Processor Scalable Family
- 12th generation Intel® Core™ processors

P6 family processors are IA-32 processors based on the P6 family microarchitecture. This includes the Pentium® Pro, Pentium® II, Pentium® III, and Pentium® III Xeon® processors.

The Pentium® 4, Pentium® D, and Pentium® processor Extreme Editions are based on the Intel NetBurst® microarchitecture. Most early Intel® Xeon® processors are based on the Intel NetBurst® microarchitecture. Intel Xeon processor 5000, 7100 series are based on the Intel NetBurst® microarchitecture.

The Intel® Core™ Duo, Intel® Core™ Solo and dual-core Intel® Xeon® processor LV are based on an improved Pentium® M processor microarchitecture.

The Intel® Xeon® processor 3000, 3200, 5100, 5300, 7200, and 7300 series, Intel® Pentium® dual-core, Intel® Core™2 Duo, Intel® Core™2 Quad, and Intel® Core™2 Extreme processors are based on Intel® Core™ microarchitecture.

The Intel® Xeon® processor 5200, 5400, 7400 series, Intel® Core™2 Quad processor Q9000 series, and Intel® Core™2 Extreme processors QX9000, X9000 series, Intel® Core™2 processor E8000 series are based on Enhanced Intel® Core™ microarchitecture.

The Intel Atom® processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are based on the Intel Atom® microarchitecture and supports Intel 64 architecture.

P6 family, Pentium® M, Intel® Core™ Solo, Intel® Core™ Duo processors, dual-core Intel® Xeon® processor LV, and early generations of Pentium 4 and Intel Xeon processors support IA-32 architecture. The Intel® Atom™ processor Z5xx series support IA-32 architecture.

The Intel® Xeon® processor 3000, 3200, 5000, 5100, 5200, 5300, 5400, 7100, 7200, 7300, 7400 series, Intel® Core™2 Duo, Intel® Core™2 Extreme, Intel® Core™2 Quad processors, Pentium® D processors, Pentium® Dual-Core processor, newer generations of Pentium 4 and Intel Xeon processor family support Intel® 64 architecture.

The Intel® Core™ i7 processor and Intel® Xeon® processor 3400, 5500, 7500 series are based on 45 nm Nehalem microarchitecture. Westmere microarchitecture is a 32 nm version of the Nehalem microarchitecture. Intel® Xeon® processor 5600 series, Intel Xeon processor E7 and various Intel Core i7, i5, i3 processors are based on the Westmere microarchitecture. These processors support Intel 64 architecture.

The Intel® Xeon® processor E5 family, Intel® Xeon® processor E3-1200 family, Intel® Xeon® processor E7-8800/4800/2800 product families, Intel® Core™ i7-3930K processor, and 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Sandy Bridge microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E7-8800/4800/2800 v2 product families, Intel® Xeon® processor E3-1200 v2 product family and 3rd generation Intel® Core™ processors are based on the Ivy Bridge microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E5-4600/2600/1600 v2 product families, Intel® Xeon® processor E5-2400/1400 v2 product families and Intel® Core™ i7-49xx Processor Extreme Edition are based on the Ivy Bridge-E microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E3-1200 v3 product family and 4th Generation Intel® Core™ processors are based on the Haswell microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E5-2600/1600 v3 product families and the Intel® Core™ i7-59xx Processor Extreme Edition are based on the Haswell-E microarchitecture and support Intel 64 architecture.

The Intel Atom® processor Z8000 series is based on the Airmont microarchitecture.

The Intel Atom® processor Z3400 series and the Intel Atom® processor Z3500 series are based on the Silvermont microarchitecture.

The Intel® Core™ M processor family, 5th generation Intel® Core™ processors, Intel® Xeon® processor D-1500 product family and the Intel® Xeon® processor E5 v4 family are based on the Broadwell microarchitecture and support Intel 64 architecture.

The Intel® Xeon® Processor Scalable Family, Intel® Xeon® processor E3-1500m v5 product family and 6th generation Intel® Core™ processors are based on the Skylake microarchitecture and support Intel 64 architecture.

The 7th generation Intel® Core™ processors are based on the Kaby Lake microarchitecture and support Intel 64 architecture.

The Intel Atom® processor C series, the Intel Atom® processor X series, the Intel® Pentium® processor J series, the Intel® Celeron® processor J series, and the Intel® Celeron® processor N series are based on the Goldmont microarchitecture.

The Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series is based on the Knights Landing microarchitecture and supports Intel 64 architecture.

The Intel® Pentium® Silver processor series, the Intel® Celeron® processor J series, and the Intel® Celeron® processor N series are based on the Goldmont Plus microarchitecture.

The 8th generation Intel® Core™ processors, 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on the Coffee Lake microarchitecture and support Intel 64 architecture.

The Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series is based on the Knights Mill microarchitecture and supports Intel 64 architecture.

The 2nd generation Intel® Xeon® Processor Scalable Family is based on the Cascade Lake product and supports Intel 64 architecture.

Some 10th generation Intel® Core™ processors are based on the Ice Lake microarchitecture, and some are based on the Comet Lake microarchitecture; both support Intel 64 architecture.

Some 11th generation Intel® Core™ processors are based on the Tiger Lake microarchitecture, and some are based on the Rocket Lake microarchitecture; both support Intel 64 architecture.

Some 3rd generation Intel® Xeon® Processor Scalable Family processors are based on the Cooper Lake product, and some are based on the Ice Lake microarchitecture; both support Intel 64 architecture.

The 12th generation Intel® Core™ processors supporting Alder Lake performance hybrid architecture support Intel 64 architecture.

IA-32 architecture is the instruction set architecture and programming environment for Intel's 32-bit microprocessors. Intel® 64 architecture is the instruction set architecture and programming environment which is the superset of Intel's 32-bit and 64-bit architectures. It is compatible with the IA-32 architecture.

1.2 OVERVIEW OF THE SYSTEM PROGRAMMING GUIDE

A description of this manual's content follows[1]:

Chapter 1 — About This Manual. Gives an overview of all volumes of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*. It also describes the notational conventions in these manuals and lists related Intel manuals and documentation of interest to programmers and hardware designers.

Chapter 2 — System Architecture Overview. Describes the modes of operation used by Intel 64 and IA-32 processors and the mechanisms provided by the architectures to support operating systems and executives, including the system-oriented registers and data structures and the system-oriented instructions. The steps necessary for switching between real-address and protected modes are also identified.

Chapter 3 — Protected-Mode Memory Management. Describes the data structures, registers, and instructions that support segmentation and paging. The chapter explains how they can be used to implement a "flat" (unsegmented) memory model or a segmented memory model.

Chapter 4 — Paging. Describes the paging modes supported by Intel 64 and IA-32 processors.

1. Model-Specific Registers have been moved out of this volume and into a separate volume: *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*.

Chapter 5 — Protection. Describes the support for page and segment protection provided in the Intel 64 and IA-32 architectures. This chapter also explains the implementation of privilege rules, stack switching, pointer validation, user and supervisor modes.

Chapter 6 — Interrupt and Exception Handling. Describes the basic interrupt mechanisms defined in the Intel 64 and IA-32 architectures, shows how interrupts and exceptions relate to protection, and describes how the architecture handles each exception type. Reference information for each exception is given in this chapter. Includes programming the LINT0 and LINT1 inputs and gives an example of how to program the LINT0 and LINT1 pins for specific interrupt vectors.

Chapter 7 — Task Management. Describes mechanisms the Intel 64 and IA-32 architectures provide to support multitasking and inter-task protection.

Chapter 8 — Multiple-Processor Management. Describes the instructions and flags that support multiple processors with shared memory, memory ordering, and Intel® Hyper-Threading Technology. Includes MP initialization for P6 family processors and gives an example of how to use the MP protocol to boot P6 family processors in an MP system.

Chapter 9 — Processor Management and Initialization. Defines the state of an Intel 64 or IA-32 processor after reset initialization. This chapter also explains how to set up an Intel 64 or IA-32 processor for real-address mode operation and protected- mode operation, and how to switch between modes.

Chapter 10 — Advanced Programmable Interrupt Controller (APIC). Describes the programming interface to the local APIC and gives an overview of the interface between the local APIC and the I/O APIC. Includes APIC bus message formats and describes the message formats for messages transmitted on the APIC bus for P6 family and Pentium processors.

Chapter 11 — Memory Cache Control. Describes the general concept of caching and the caching mechanisms supported by the Intel 64 or IA-32 architectures. This chapter also describes the memory type range registers (MTRRs) and how they can be used to map memory types of physical memory. Information on using the new cache control and memory streaming instructions introduced with the Pentium III, Pentium 4, and Intel Xeon processors is also given.

Chapter 12 — Intel® MMX™ Technology System Programming. Describes those aspects of the Intel® MMX™ technology that must be handled and considered at the system programming level, including: task switching, exception handling, and compatibility with existing system environments.

Chapter 13 — System Programming For Instruction Set Extensions And Processor Extended States. Describes the operating system requirements to support SSE/SSE2/SSE3/SSSE3/SSE4 extensions, including task switching, exception handling, and compatibility with existing system environments. The latter part of this chapter describes the extensible framework of operating system requirements to support processor extended states. Processor extended state may be required by instruction set extensions beyond those of SSE/SSE2/SSE3/SSSE3/SSE4 extensions.

Chapter 14 — Power and Thermal Management. Describes facilities of Intel 64 and IA-32 architecture used for power management and thermal monitoring.

Chapter 15 — Machine-Check Architecture. Describes the machine-check architecture and machine-check exception mechanism found in the Pentium 4, Intel Xeon, and P6 family processors. Additionally, a signaling mechanism for software to respond to hardware corrected machine check error is covered.

Chapter 16 — Interpreting Machine-Check Error Codes. Gives an example of how to interpret the error codes for a machine-check error that occurred on a P6 family processor.

Chapter 17 — Debug, Branch Profile, TSC, and Resource Monitoring Features. Describes the debugging registers and other debug mechanism provided in Intel 64 or IA-32 processors. This chapter also describes the time-stamp counter.

Chapter 18 — Last Branch Records. Describes the Last Branch Records (architectural feature).

Chapter 19 — Performance Monitoring. Describes the Intel 64 and IA-32 architectures' facilities for monitoring performance.

Chapter 20 — 8086 Emulation. Describes the real-address and virtual-8086 modes of the IA-32 architecture.

Chapter 21 — Mixing 16-Bit and 32-Bit Code. Describes how to mix 16-bit and 32-bit code modules within the same program or task.

Chapter 22 — IA-32 Architecture Compatibility. Describes architectural compatibility among IA-32 processors.

Chapter 23 — Introduction to Virtual Machine Extensions. Describes the basic elements of virtual machine architecture and the virtual machine extensions for Intel 64 and IA-32 Architectures.

Chapter 24 — Virtual Machine Control Structures. Describes components that manage VMX operation. These include the working-VMCS pointer and the controlling-VMCS pointer.

Chapter 25 — VMX Non-Root Operation. Describes the operation of a VMX non-root operation. Processor operation in VMX non-root mode can be restricted programmatically such that certain operations, events or conditions can cause the processor to transfer control from the guest (running in VMX non-root mode) to the monitor software (running in VMX root mode).

Chapter 26 — VM Entries. Describes VM entries. VM entry transitions the processor from the VMM running in VMX root-mode to a VM running in VMX non-root mode. VM-Entry is performed by the execution of VMLAUNCH or VMRESUME instructions.

Chapter 27 — VM Exits. Describes VM exits. Certain events, operations or situations while the processor is in VMX non-root operation may cause VM-exit transitions. In addition, VM exits can also occur on failed VM entries.

Chapter 28 — VMX Support for Address Translation. Describes virtual-machine extensions that support address translation and the virtualization of physical memory.

Chapter 29 — APIC Virtualization and Virtual Interrupts. Describes the VMCS including controls that enable the virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC).

Chapter 30 — VMX Instruction Reference. Describes the virtual-machine extensions (VMX). VMX is intended for a system executive to support virtualization of processor hardware and a system software layer acting as a host to multiple guest software environments.

Chapter 31 — System Management Mode. Describes Intel 64 and IA-32 architectures' system management mode (SMM) facilities.

Chapter 32 — Intel® Processor Trace. Describes details of Intel® Processor Trace.

Chapter 33 — Introduction to Intel® Software Guard Extensions. Provides an overview of the Intel® Software Guard Extensions (Intel® SGX) set of instructions.

Chapter 34 — Enclave Access Control and Data Structures. Describes Enclave Access Control procedures and defines various Intel SGX data structures.

Chapter 35 — Enclave Operation. Describes enclave creation and initialization, adding pages and measuring an enclave, and enclave entry and exit.

Chapter 36 — Enclave Exiting Events. Describes enclave-exiting events (EEE) and asynchronous enclave exit (AEX).

Chapter 37 — SGX Instruction References. Describes the supervisor and user level instructions provided by Intel SGX.

Chapter 38 — Intel® SGX Interactions with IA32 and Intel® 64 Architecture. Describes the Intel SGX collection of enclave instructions for creating protected execution environments on processors supporting IA32 and Intel 64 architectures.

Chapter 39 — Enclave Code Debug and Profiling. Describes enclave code debug processes and options.

Appendix A — VMX Capability Reporting Facility. Describes the VMX capability MSRs. Support for specific VMX features is determined by reading capability MSRs.

Appendix B — Field Encoding in VMCS. 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.).

Appendix C — VM Basic Exit Reasons. Describes the 32-bit fields that encode reasons for a VM exit. Examples of exit reasons include, but are not limited to: software interrupts, processor exceptions, software traps, NMIs, external interrupts, and triple faults.

1.3 NOTATIONAL CONVENTIONS

This manual uses specific notation for data-structure formats, for symbolic representation of instructions, and for hexadecimal and binary numbers. A review of this notation makes the manual easier to read.

1.3.1 Bit and Byte Order

In illustrations of data structures in memory, smaller addresses appear toward the bottom of the figure; addresses increase toward the top. Bit positions are numbered from right to left. The numerical value of a set bit is equal to two raised to the power of the bit position. Intel 64 and IA-32 processors are "little endian" machines; this means the bytes of a word are numbered starting from the least significant byte. Figure 1-1 illustrates these conventions.

1.3.2 Reserved Bits and Software Compatibility

In many register and memory layout descriptions, certain bits are marked as **reserved**. When bits are marked as reserved, it is essential for compatibility with future processors that software treat these bits as having a future, though unknown, effect. The behavior of reserved bits should be regarded as not only undefined, but unpredictable. Software should follow these guidelines in dealing with reserved bits:

- Do not depend on the states of any reserved bits when testing the values of registers which contain such bits. Mask out the reserved bits before testing.
- Do not depend on the states of any reserved bits when storing to memory or to a register.
- Do not depend on the ability to retain information written into any reserved bits.
- When loading a register, always load the reserved bits with the values indicated in the documentation, if any, or reload them with values previously read from the same register.

NOTE

Avoid any software dependence upon the state of reserved bits in Intel 64 and IA-32 registers. Depending upon the values of reserved register bits will make software dependent upon the unspecified manner in which the processor handles these bits. Programs that depend upon reserved values risk incompatibility with future processors.

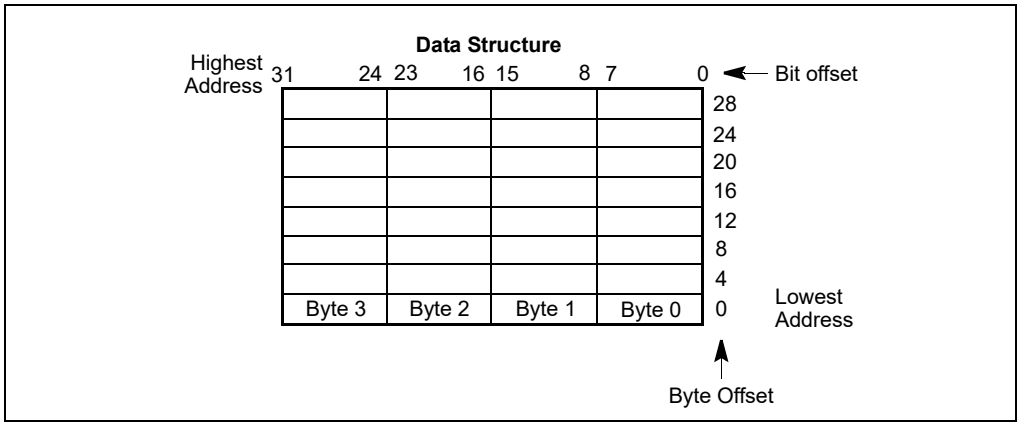


Figure 1-1. Bit and Byte Order

1.3.3 Instruction Operands

When instructions are represented symbolically, a subset of assembly language is used. In this subset, an instruction has the following format:

```
label: mnemonic argument1, argument2, argument3
```

where:

- A **label** is an identifier which is followed by a colon.
- A **mnemonic** is a reserved name for a class of instruction opcodes which have the same function.
- The operands **argument1**, **argument2**, and **argument3** are optional. There may be from zero to three operands, depending on the opcode. When present, they take the form of either literals or identifiers for data items. Operand identifiers are either reserved names of registers or are assumed to be assigned to data items declared in another part of the program (which may not be shown in the example).

When two operands are present in an arithmetic or logical instruction, the right operand is the source and the left operand is the destination.

For example:

```
LOADREG: MOV EAX, SUBTOTAL
```

In this example LOADREG is a label, MOV is the mnemonic identifier of an opcode, EAX is the destination operand, and SUBTOTAL is the source operand. Some assembly languages put the source and destination in reverse order.

1.3.4 Hexadecimal and Binary Numbers

Base 16 (hexadecimal) numbers are represented by a string of hexadecimal digits followed by the character H (for example, F82EH). A hexadecimal digit is a character from the following set: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Base 2 (binary) numbers are represented by a string of 1s and 0s, sometimes followed by the character B (for example, 1010B). The "B" designation is only used in situations where confusion as to the type of number might arise.

1.3.5 Segmented Addressing

The processor uses byte addressing. This means memory is organized and accessed as a sequence of bytes. Whether one or more bytes are being accessed, a byte address is used to locate the byte or bytes memory. The range of memory that can be addressed is called an **address space**.

The processor also supports segmented addressing. This is a form of addressing where a program may have many independent address spaces, called **segments**. For example, a program can keep its code (instructions) and stack in separate segments. Code addresses would always refer to the code space, and stack addresses would always refer to the stack space. The following notation is used to specify a byte address within a segment:

Segment-register:Byte-address

For example, the following segment address identifies the byte at address FF79H in the segment pointed by the DS register:

```
DS:FF79H
```

The following segment address identifies an instruction address in the code segment. The CS register points to the code segment and the EIP register contains the address of the instruction.

```
CS:EIP
```

1.3.6 Syntax for CPUID, CR, and MSR Values

Obtain feature flags, status, and system information by using the CPUID instruction, by checking control register bits, and by reading model-specific registers. We are moving toward a single syntax to represent this type of information. See Figure 1-2.

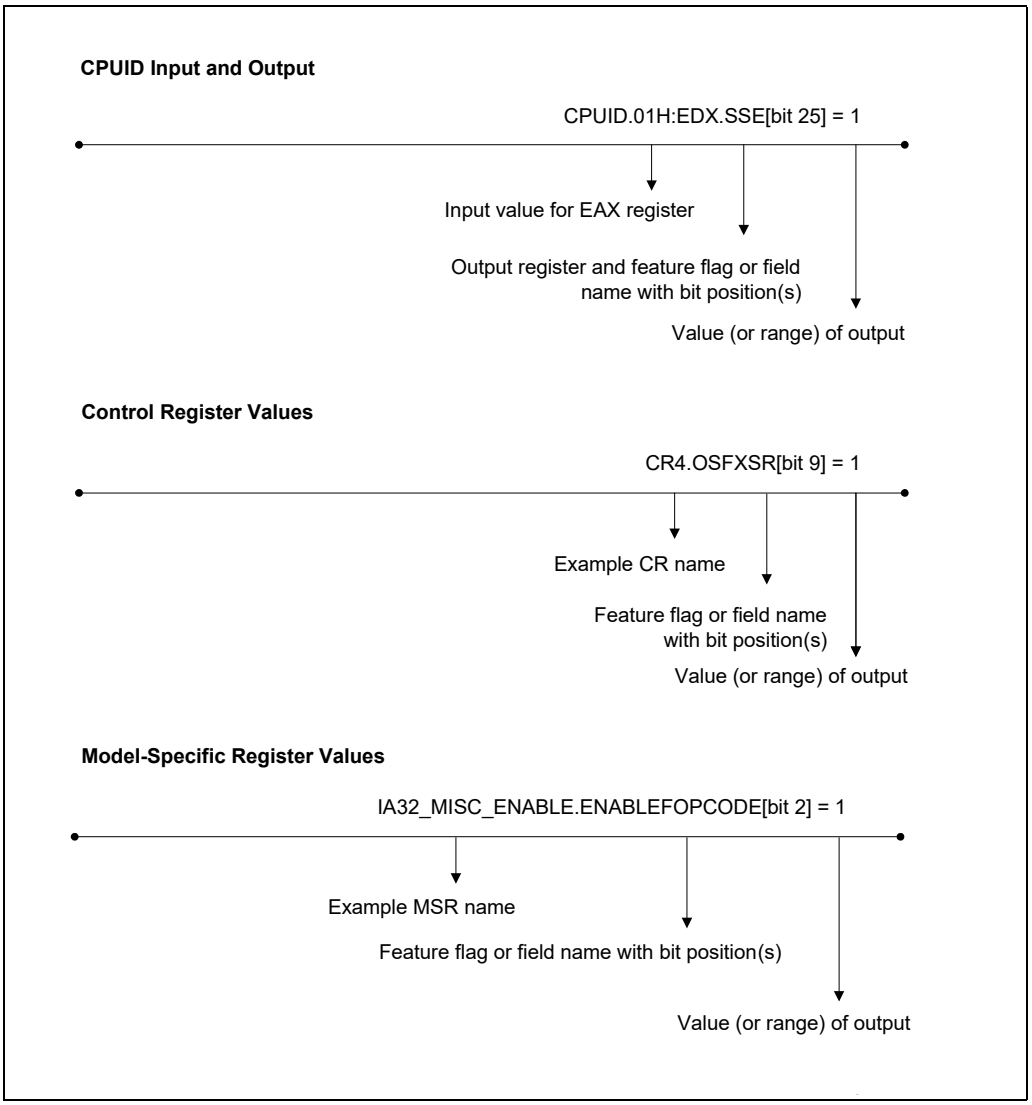


Figure 1-2. Syntax for CPUID, CR, and MSR Data Presentation

1.3.7 Exceptions

An exception is an event that typically occurs when an instruction causes an error. For example, an attempt to divide by zero generates an exception. However, some exceptions, such as breakpoints, occur under other conditions. Some types of exceptions may provide error codes. An error code reports additional information about the error. An example of the notation used to show an exception and error code is shown below:

```
#PF(fault code)
```

This example refers to a page-fault exception under conditions where an error code naming a type of fault is reported. Under some conditions, exceptions which produce error codes may not be able to report an accurate code. In this case, the error code is zero, as shown below for a general-protection exception:

```
#GP(0)
```

1.4 RELATED LITERATURE

Literature related to Intel 64 and IA-32 processors is listed and viewable on-line at:

https://software.intel.com/en-us/articles/intel-sdm

See also:

- The latest security information on Intel® products:
  https://www.intel.com/content/www/us/en/security-center/default.html
- Software developer resources, guidance and insights for security advisories:
  https://software.intel.com/security-software-guidance/
- The data sheet for a particular Intel 64 or IA-32 processor
- The specification update for a particular Intel 64 or IA-32 processor
- Intel® C++ Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Fortran Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Software Development Tools:
  https://software.intel.com/en-us/intel-sdp-home
- Intel® 64 and IA-32 Architectures Software Developer's Manual (in one, four or ten volumes):
  https://software.intel.com/en-us/articles/intel-sdm
- Intel® 64 and IA-32 Architectures Optimization Reference Manual:
  https://software.intel.com/en-us/articles/intel-sdm#optimization
- Intel® Trusted Execution Technology Measured Launched Environment Programming Guide:
  http://www.intel.com/content/www/us/en/software-developers/intel-txt-software-development-guide.html
- Intel® Software Guard Extensions (Intel® SGX) Information
  https://software.intel.com/en-us/isa-extensions/intel-sgx
- Developing Multi-threaded Applications: A Platform Consistent Approach:
  https://software.intel.com/sites/default/files/article/147714/51534-developing-multithreaded-applications.pdf
- Using Spin-Loops on Intel® Pentium® 4 Processor and Intel® Xeon® Processor:
  https://software.intel.com/sites/default/files/22/30/25602
- Performance Monitoring Unit Sharing Guide
  http://software.intel.com/file/30388

Literature related to select features in future Intel processors are available at:

- Intel® Architecture Instruction Set Extensions Programming Reference
  https://software.intel.com/en-us/isa-extensions

More relevant links are:

- Intel® Developer Zone:
  https://software.intel.com/en-us
- Developer centers:
  http://www.intel.com/content/www/us/en/hardware-developers/developer-centers.html
- Processor support general link:
  http://www.intel.com/support/processors/
- Intel® Hyper-Threading Technology (Intel® HT Technology):
  http://www.intel.com/technology/platform-technology/hyper-threading/index.htm

11. Updates to Chapter 4, Volume 3A

Change bars and green text show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

Changes to this chapter: HLAT updates made throughout chapter.

CHAPTER 4
PAGING

Chapter 3 explains how segmentation converts logical addresses to linear addresses. **Paging** (or linear-address translation) is the process of translating linear addresses so that they can be used to access memory or I/O devices. Paging translates each linear address to a **physical address** and determines, for each translation, what accesses to the linear address are allowed (the address's **access rights**) and the type of caching used for such accesses (the address's **memory type**).

Intel-64 processors support four different paging modes. These modes are identified and defined in Section 4.1. Section 4.2 gives an overview of the translation mechanism that is used in all modes. Section 4.3, Section 4.4, and Section 4.5 discuss the four paging modes in detail.

Section 4.6 details how paging determines and uses access rights. Section 4.7 discusses exceptions that may be generated by paging (page-fault exceptions). Section 4.8 considers data which the processor writes in response to linear-address accesses (accessed and dirty flags).

Section 4.9 describes how paging determines the memory types used for accesses to linear addresses. Section 4.10 provides details of how a processor may cache information about linear-address translation. Section 4.11 outlines interactions between paging and certain VMX features. Section 4.12 gives an overview of how paging can be used to implement virtual memory.

4.1 PAGING MODES AND CONTROL BITS

Paging behavior is controlled by the following control bits:

- The WP and PG flags in control register CR0 (bit 16 and bit 31, respectively).
- The PSE, PAE, PGE, LA57, PCIDE, SMEP, SMAP, PKE, CET, and PKS flags in control register CR4 (bit 4, bit 5, bit 7, bit 12, bit 17, bit 20, bit 21, bit 22, bit 23, and bit 24, respectively).
- The LME and NXE flags in the IA32_EFER MSR (bit 8 and bit 11, respectively).
- The AC flag in the EFLAGS register (bit 18).
- The "enable HLAT" VM-execution control (tertiary processor-based VM-execution control bit 1; see Section 24.6.2, "Processor-Based VM-Execution Controls," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*).

Software enables paging by using the MOV to CR0 instruction to set CR0.PG. Before doing so, software should ensure that control register CR3 contains the physical address of the first paging structure that the processor will use for linear-address translation (see Section 4.2) and that that structure is initialized as desired. See Table 4-3, Table 4-7, and Table 4-12 for the use of CR3 in the different paging modes.

Section 4.1.1 describes how the values of CR0.PG, CR4.PAE, CR4.LA57, and IA32_EFER.LME determine whether paging is enabled and, if so, which of four paging modes is in use. Section 4.1.2 explains how to manage these bits to establish or make changes in paging modes. Section 4.1.3 discusses how CR0.WP, CR4.PSE, CR4.PGE, CR4.PCIDE, CR4.SMEP, CR4.SMAP, CR4.PKE, CR4.CET, CR4.PKS, and IA32_EFER.NXE modify the operation of the different paging modes.

4.1.1 Four Paging Modes

If CR0.PG = 0, paging is not used. The logical processor treats all linear addresses as if they were physical addresses. CR4.PAE, CR4.LA57, and IA32_EFER.LME are ignored by the processor, as are CR0.WP, CR4.PSE, CR4.PGE, CR4.SMEP, CR4.SMAP, and IA32_EFER.NXE. (CR4.CET is also ignored insofar as it affects linear-address access rights.)

Paging is enabled if CR0.PG = 1. Paging can be enabled only if protection is enabled (CR0.PE = 1). If paging is enabled, one of four paging modes is used. The values of CR4.PAE, CR4.LA57, and IA32_EFER.LME determine which paging mode is used:

- If CR4.PAE = 0, **32-bit paging** is used. 32-bit paging is detailed in Section 4.3. 32-bit paging uses CR0.WP, CR4.PSE, CR4.PGE, CR4.SMEP, CR4.SMAP, and CR4.CET as described in Section 4.1.3 and Section 4.6.
- If CR4.PAE = 1 and IA32_EFER.LME = 0, **PAE paging** is used. PAE paging is detailed in Section 4.4. PAE paging uses CR0.WP, CR4.PGE, CR4.SMEP, CR4.SMAP, CR4.CET, and IA32_EFER.NXE as described in Section 4.1.3 and Section 4.6.
- If CR4.PAE = 1, IA32_EFER.LME = 1, and CR4.LA57 = 0, **4-level paging**[1] is used.[2] 4-level paging is detailed in Section 4.5 (along with 5-level paging). 4-level paging uses CR0.WP, CR4.PGE, CR4.PCIDE, CR4.SMEP, CR4.SMAP, CR4.PKE, CR4.CET, CR4.PKS, and IA32_EFER.NXE as described in Section 4.1.3 and Section 4.6.
- If CR4.PAE = 1, IA32_EFER.LME = 1, and CR4.LA57 = 1, **5-level paging** is used. 5-level paging is detailed in Section 4.5 (along with 4-level paging). 5-level paging uses CR0.WP, CR4.PGE, CR4.PCIDE, CR4.SMEP, CR4.SMAP, CR4.PKE, CR4.CET, CR4.PKS, and IA32_EFER.NXE as described in Section 4.1.3 and Section 4.6.

NOTE

32-bit paging and PAE paging can be used only in legacy protected mode (IA32_EFER.LME = 0). In contrast, 4-level paging and 5-level paging can be used only IA-32e mode (IA32_EFER.LME = 1).

The four paging modes differ with regard to the following details:

- Linear-address width. The size of the linear addresses that can be translated.
- Physical-address width. The size of the physical addresses produced by paging.
- Page size. The granularity at which linear addresses are translated. Linear addresses on the same page are translated to corresponding physical addresses on the same page.
- Support for execute-disable access rights. In some paging modes, software can be prevented from fetching instructions from pages that are otherwise readable.
- Support for PCIDs. With 4-level paging and 5-level paging, software can enable a facility by which a logical processor caches information for multiple linear-address spaces. The processor may retain cached information when software switches between different linear-address spaces.
- Support for protection keys. With 4-level paging and 5-level paging, each linear address is associated with a **protection key**. Software can use the protection-key rights registers to disable, for each protection key, how certain accesses to linear addresses associated with that protection key.

Table 4-1 illustrates the principal differences between the four paging modes.

Table 4-1. Properties of Different Paging Modes

Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	LA57 in CR4	Lin.-Addr. Width	Phys.-Addr. Width[1]	Page Sizes	Supports Execute-Disable?	Supports PCIDs and protection keys?
None	0	N/A	N/A	N/A	32	32	N/A	No	No
32-bit	1	0	0[2]	N/A	32	Up to 40[3]	4 KB 4 MB[4]	No	No
PAE	1	1	0	N/A	32	Up to 52	4 KB 2 MB	Yes[5]	No
4-level	1	1	1	0	48	Up to 52	4 KB 2 MB 1 GB[6]	Yes[5]	Yes[7]

1. Earlier versions of this manual used the term "IA-32e paging" to identify 4-level paging.
2. The LMA flag in the IA32_EFER MSR (bit 10) is a status bit that indicates whether the logical processor is in IA-32e mode (and thus uses either 4-level paging or 5-level paging). The processor always sets IA32_EFER.LMA to CR0.PG & IA32_EFER.LME. Software cannot directly modify IA32_EFER.LMA; an execution of WRMSR to the IA32_EFER MSR ignores bit 10 of its source operand.

Table 4-1. Properties of Different Paging Modes (Contd.)

Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	LA57 in CR4	Lin.-Addr. Width	Phys.-Addr. Width[1]	Page Sizes	Supports Execute-Disable?	Supports PCIDs and protection keys?
5-level	1	1	1	1	57	Up to 52	4 KB 2 MB 1 GB[6]	Yes[5]	Yes[7]

NOTES:

1. The physical-address width is always bounded by MAXPHYADDR; see Section 4.1.4.
2. The processor ensures that IA32_EFER.LME must be 0 if CR0.PG = 1 and CR4.PAE = 0.
3. 32-bit paging supports physical-address widths of more than 32 bits only for 4-MByte pages and only if the PSE-36 mechanism is supported; see Section 4.1.4 and Section 4.3.
4. 32-bit paging uses 4-MByte pages only if CR4.PSE = 1; see Section 4.3.
5. Execute-disable access rights are applied only if IA32_EFER.NXE = 1; see Section 4.6.
6. Processors that support 4-level paging or 5-level paging do not necessarily support 1-GByte pages; see Section 4.1.4.
7. PCIDs are used only if CR4.PCIDE = 1; see Section 4.10.1. Protection keys are used only if certain conditions hold; see Section 4.6.2.

Because 32-bit paging and PAE paging are used only in legacy protected mode and because legacy protected mode cannot produce linear addresses larger than 32 bits, 32-bit paging and PAE paging translate 32-bit linear addresses.

4-level paging and 5-level paging are used only in IA-32e mode. IA-32e mode has two sub-modes:

- Compatibility mode. This sub-mode uses only 32-bit linear addresses. In this sub-mode, 4-level paging and 5-level paging treat bits 63:32 of such an address as all 0.
- 64-bit mode. While this sub-mode produces 64-bit linear addresses, the processor enforces **canonicality**, meaning that the upper bits of such an address are identical: bits 63:47 for 4-level paging and bits 63:56 for 5-level paging. 4-level paging (respectively, 5-level paging) does not use bits 63:48 (respectively, bits 63:57) of such addresses.

4.1.2 Paging-Mode Enabling

If CR0.PG = 1, a logical processor is in one of four paging modes, depending on the values of CR4.PAE, IA32_EFER.LME, and CR4.LA57. Figure 4-1 illustrates how software can enable these modes and make transitions between them. The following items identify certain limitations and other details:

- IA32_EFER.LME cannot be modified while paging is enabled (CR0.PG = 1). Attempts to do so using WRMSR cause a general-protection exception (#GP(0)).
- Paging cannot be enabled (by setting CR0.PG to 1) while CR4.PAE = 0 and IA32_EFER.LME = 1. Attempts to do so using MOV to CR0 cause a general-protection exception (#GP(0)).
- One node in Figure 4-1 is labeled "IA-32e mode." This node represents either 4-level paging (if CR4.LA57 = 0) or 5-level paging (if CR4.LA57 = 1). As noted in the following items, software cannot modify CR4.LA57 (effecting transition between 4-level paging and 5-level paging) without first disabling paging.
- CR4.PAE and CR4.LA57 cannot be modified while either 4-level paging or 5-level paging is in use (when CR0.PG = 1 and IA32_EFER.LME = 1). Attempts to do so using MOV to CR4 cause a general-protection exception (#GP(0)).
- Regardless of the current paging mode, software can disable paging by clearing CR0.PG with MOV to CR0.[1]
- Software can transition between 32-bit paging and PAE paging by changing the value of CR4.PAE with MOV to CR4.

1. If the logical processor is in 64-bit mode or if CR4.PCIDE = 1, an attempt to clear CR0.PG causes a general-protection exception (#GP). Software should transition to compatibility mode and clear CR4.PCIDE before attempting to disable paging.

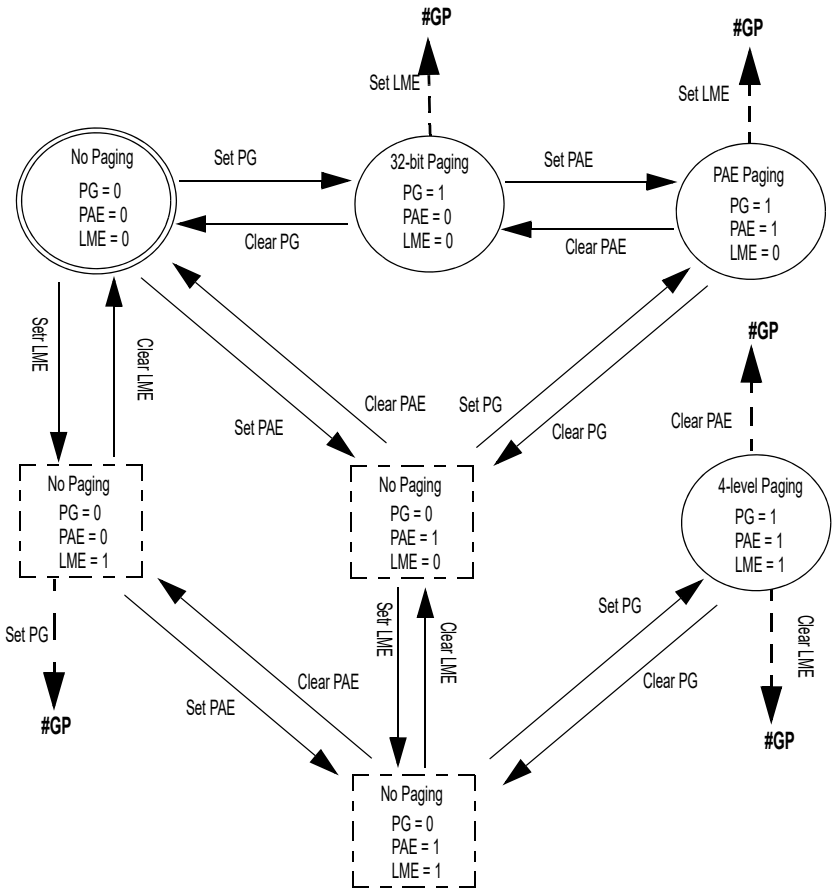


Figure 4-1. Enabling and Changing Paging Modes

- Software cannot transition directly between 4-level paging (or 5-level paging) and any of other paging mode. It must first disable paging (by clearing CR0.PG with MOV to CR0), then set CR4.PAE, IA32_EFER.LME, and CR4.LA57 to the desired values (with MOV to CR4 and WRMSR), and then re-enable paging (by setting CR0.PG with MOV to CR0). As noted earlier, an attempt to modify CR4.PAE, IA32_EFER.LME, or CR.LA57 while 4-level paging or 5-level paging is enabled causes a general-protection exception (#GP(0)).
- VMX transitions allow transitions between paging modes that are not possible using MOV to CR or WRMSR. This is because VMX transitions can load CR0, CR4, and IA32_EFER in one operation. See Section 4.11.1.

4.1.3 Paging-Mode Modifiers

Details of how each paging mode operates are determined by the following control bits:

- The WP flag in CR0 (bit 16).
- The PSE, PGE, PCIDE, SMEP, SMAP, PKE, CET, and PKS flags in CR4 (bit 4, bit 7, bit 17, bit 20, bit 21, bit 22, bit 23, and bit 24, respectively).
- The NXE flag in the IA32_EFER MSR (bit 11).
- The “enable HLAT” VM-execution control (tertiary processor-based VM-execution control bit 1).

CR0.WP allows pages to be protected from supervisor-mode writes. If CR0.WP = 0, supervisor-mode write accesses are allowed to linear addresses with read-only access rights; if CR0.WP = 1, they are not. (User-mode write accesses are never allowed to linear addresses with read-only access rights, regardless of the value of

CR0.WP.) Section 4.6 explains how access rights are determined, including the definition of supervisor-mode and user-mode accesses.

CR4.PSE enables 4-MByte pages for 32-bit paging. If CR4.PSE = 0, 32-bit paging can use only 4-KByte pages; if CR4.PSE = 1, 32-bit paging can use both 4-KByte pages and 4-MByte pages. See Section 4.3 for more information. (PAE paging, 4-level paging, and 5-level paging can use multiple page sizes regardless of the value of CR4.PSE.)

CR4.PGE enables global pages. If CR4.PGE = 0, no translations are shared across address spaces; if CR4.PGE = 1, specified translations may be shared across address spaces. See Section 4.10.2.4 for more information.

CR4.PCIDE enables process-context identifiers (PCIDs) for 4-level paging and 5-level paging. PCIDs allow a logical processor to cache information for multiple linear-address spaces. See Section 4.10.1 for more information.

CR4.SMEP allows pages to be protected from supervisor-mode instruction fetches. If CR4.SMEP = 1, software operating in supervisor mode cannot fetch instructions from linear addresses that are accessible in user mode. Section 4.6 explains how access rights are determined, including the definition of supervisor-mode accesses and user-mode accessibility.

CR4.SMAP allows pages to be protected from supervisor-mode data accesses. If CR4.SMAP = 1, software operating in supervisor mode cannot access data at linear addresses that are accessible in user mode. Software can override this protection by setting EFLAGS.AC. Section 4.6 explains how access rights are determined, including the definition of supervisor-mode accesses and user-mode accessibility.

CR4.PKE and CR4.PKS enable specification of access rights based on **protection keys**. 4-level paging and 5-level paging associate each linear address with a protection key. When CR4.PKE = 1, the PKRU register specifies, for each protection key, whether user-mode linear addresses with that protection key can be read or written. When CR4.PKS = 1, the IA32_PKRS MSR does the same for supervisor-mode linear addresses. See Section 4.6 for more information.

CR4.CET enables **control-flow enforcement technology**, including the shadow-stack feature. If CR4.CET = 1, certain memory accesses are identified as **shadow-stack accesses** and certain linear addresses translate to **shadow-stack pages**. Section 4.6 explains how access rights are determined for these accesses and pages. (The processor allows CR4.CET to be set only if CR0.WP is also set.)

IA32_EFER.NXE enables execute-disable access rights for PAE paging, 4-level paging, and 5-level paging. If IA32_EFER.NXE = 1, instruction fetches can be prevented from specified linear addresses (even if data reads from the addresses are allowed). Section 4.6 explains how access rights are determined. (IA32_EFER.NXE has no effect with 32-bit paging. Software that wants to use this feature to limit instruction fetches from readable pages must use PAE paging, 4-level paging, or 5-level paging.)

The "enable HLAT" VM-execution control enables **HLAT paging** for 4-level paging and 5-level paging. HLAT paging does not use control register CR3 to identify the address of the first paging structure used for linear-address translation; instead, that structure is located using a field in the virtual-machine control structure (VMCS). In addition, HLAT paging interprets certain bits in paging-structure entries differently than ordinary paging. See Section 4.5 for details.

4.1.4 Enumeration of Paging Features by CPUID

Software can discover support for different paging features using the CPUID instruction:

- PSE: page-size extensions for 32-bit paging.
  If CPUID.01H:EDX.PSE [bit 3] = 1, CR4.PSE may be set to 1, enabling support for 4-MByte pages with 32-bit paging (see Section 4.3).
- PAE: physical-address extension.
  If CPUID.01H:EDX.PAE [bit 6] = 1, CR4.PAE may be set to 1, enabling PAE paging (this setting is also required for 4-level paging and 5-level paging).
- PGE: global-page support.
  If CPUID.01H:EDX.PGE [bit 13] = 1, CR4.PGE may be set to 1, enabling the global-page feature (see Section 4.10.2.4).
- PAT: page-attribute table.
  If CPUID.01H:EDX.PAT [bit 16] = 1, the 8-entry page-attribute table (PAT) is supported. When the PAT is

supported, three bits in certain paging-structure entries select a memory type (used to determine type of caching used) from the PAT (see Section 4.9.2).

- PSE-36: page-size extensions with 40-bit physical-address extension.
  If CPUID.01H:EDX.PSE-36 [bit 17] = 1, the PSE-36 mechanism is supported, indicating that translations using 4-MByte pages with 32-bit paging may produce physical addresses with up to 40 bits (see Section 4.3).
- PCID: process-context identifiers.
  If CPUID.01H:ECX.PCID [bit 17] = 1, CR4.PCIDE may be set to 1, enabling process-context identifiers (see Section 4.10.1).
- SMEP: supervisor-mode execution prevention.
  If CPUID.(EAX=07H,ECX=0H):EBX.SMEP [bit 7] = 1, CR4.SMEP may be set to 1, enabling supervisor-mode execution prevention (see Section 4.6).
- SMAP: supervisor-mode access prevention.
  If CPUID.(EAX=07H,ECX=0H):EBX.SMAP [bit 20] = 1, CR4.SMAP may be set to 1, enabling supervisor-mode access prevention (see Section 4.6).
- PKU: protection keys for user-mode pages.
  If CPUID.(EAX=07H,ECX=0H):ECX.PKU [bit 3] = 1, CR4.PKE may be set to 1, enabling protection keys for user-mode pages (see Section 4.6).
- OSPKE: enabling of protection keys for user-mode pages.
  CPUID.(EAX=07H,ECX=0H):ECX.OSPKE [bit 4] returns the value of CR4.PKE. Thus, protection keys for user-mode pages are enabled if this flag is 1 (see Section 4.6).
- CET: control-flow enforcement technology.
  If CPUID.(EAX=07H,ECX=0H):ECX.CET_SS [bit 7] = 1, CR4.CET may be set to 1, enabling shadow-stack pages (see Section 4.6).
- LA57: 57-bit linear addresses and 5-level paging.
  If CPUID.(EAX=07H,ECX=0):ECX.LA57 [bit 16] = 1, CR4.LA57 may be set to 1, enabling 5-level paging.
- PKS: protection keys for supervisor-mode pages.
  If CPUID.(EAX=07H,ECX=0H):ECX.PKS [bit 31] = 1, CR4.PKS may be set to 1, enabling protection keys for supervisor-mode pages (see Section 4.6).
- NX: execute disable.
  If CPUID.80000001H:EDX.NX [bit 20] = 1, IA32_EFER.NXE may be set to 1, allowing software to disable execute access to selected pages (see Section 4.6). (Processors that do not support CPUID function 80000001H do not allow IA32_EFER.NXE to be set to 1.)
- Page1GB: 1-GByte pages.
  If CPUID.80000001H:EDX.Page1GB [bit 26] = 1, 1-GByte pages may be supported with 4-level paging and 5-level paging (see Section 4.5).
- LM: IA-32e mode support.
  If CPUID.80000001H:EDX.LM [bit 29] = 1, IA32_EFER.LME may be set to 1, enabling IA-32e mode (with either 4-level paging or 5-level paging). (Processors that do not support CPUID function 80000001H do not allow IA32_EFER.LME to be set to 1.)
- CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. (For processors that do not support CPUID function 80000008H, the width is generally 36 if CPUID.01H:EDX.PAE [bit 6] = 1 and 32 otherwise.) This width is referred to as MAXPHYADDR. MAXPHYADDR is at most 52.
- CPUID.80000008H:EAX[15:8] reports the linear-address width supported by the processor. Generally, this value is reported as follows:
  - If CPUID.80000001H:EDX.LM [bit 29] = 0, the value is reported as 32.
  - If CPUID.80000001H:EDX.LM [bit 29] = 1 and CPUID.(EAX=07H,ECX=0):ECX.LA57 [bit 16] = 0, the value is reported as 48.
  - If CPUID.(EAX=07H,ECX=0):ECX.LA57 [bit 16] = 1, the value is reported as 57.

  (Processors that do not support CPUID function 80000008H, support a linear-address width of 32.)

4.2 HIERARCHICAL PAGING STRUCTURES: AN OVERVIEW

All four paging modes translate linear addresses using **hierarchical paging structures**. This section provides an overview of their operation. Section 4.3, Section 4.4, Section 4.5, and Section 4.6 provide details for the four paging modes.

Every paging structure is 4096 Bytes in size and comprises a number of individual **entries**. With 32-bit paging, each entry is 32 bits (4 bytes); there are thus 1024 entries in each structure. With the other paging modes, each entry is 64 bits (8 bytes); there are thus 512 entries in each structure. (PAE paging includes one exception, a paging structure that is 32 bytes in size, containing 4 64-bit entries.)

The processor uses the upper portion of a linear address to identify a series of paging-structure entries. The last of these entries identifies the physical address of the region to which the linear address translates (called the **page frame**). The lower portion of the linear address (called the **page offset**) identifies the specific address within that region to which the linear address translates.

Each paging-structure entry contains a physical address, which is either the address of another paging structure or the address of a page frame. In the first case, the entry is said to **reference** the other paging structure; in the latter, the entry is said to **map a page**.

The first paging structure used for any translation is located at the physical address in CR3.[1] A linear address is translated using the following iterative procedure. A portion of the linear address (initially the uppermost bits) selects an entry in a paging structure (initially the one located using CR3). If that entry references another paging structure, the process continues with that paging structure and with the portion of the linear address immediately below that just used. If instead the entry maps a page, the process completes: the physical address in the entry is that of the page frame and the remaining lower portion of the linear address is the page offset.

The following items give an example for each of the four paging modes (each example locates a 4-KByte page frame):

- With 32-bit paging, each paging structure comprises $1024 = 2^{10}$ entries. For this reason, the translation process uses 10 bits at a time from a 32-bit linear address. Bits 31:22 identify the first paging-structure entry and bits 21:12 identify a second. The latter identifies the page frame. Bits 11:0 of the linear address are the page offset within the 4-KByte page frame. (See Figure 4-2 for an illustration.)
- With PAE paging, the first paging structure comprises only $4 = 2^2$ entries. Translation thus begins by using bits 31:30 from a 32-bit linear address to identify the first paging-structure entry. Other paging structures comprise $512 = 2^9$ entries, so the process continues by using 9 bits at a time. Bits 29:21 identify a second paging-structure entry and bits 20:12 identify a third. This last identifies the page frame. (See Figure 4-5 for an illustration.)
- With 4-level paging, each paging structure comprises $512 = 2^9$ entries and translation uses 9 bits at a time from a 48-bit linear address. Bits 47:39 identify the first paging-structure entry, bits 38:30 identify a second, bits 29:21 a third, and bits 20:12 identify a fourth. Again, the last identifies the page frame. (See Figure 4-8 for an illustration.)
- 5-level paging is similar to 4-level paging except that 5-level paging translates 57-bit linear addresses. Bits 56:48 identify the first paging-structure entry, while the remaining bits are used as with 4-level paging.

The translation process in each of the examples above completes by identifying a page frame; the page frame is part of the **translation** of the original linear address. In some cases, however, the paging structures may be configured so that the translation process terminates before identifying a page frame. This occurs if the process encounters a paging-structure entry that is marked "not present" (because its P flag — bit 0 — is clear) or in which a reserved bit is set. In this case, there is no translation for the linear address; an access to that address causes a page-fault exception (see Section 4.7).

In the examples above, a paging-structure entry maps a page with a 4-KByte page frame when only 12 bits remain in the linear address; entries identified earlier always reference other paging structures. That may not apply in other cases. The following items identify when an entry maps a page and when it references another paging structure:

- If more than 12 bits remain in the linear address, bit 7 (PS — page size) of the current paging-structure entry is consulted. If the bit is 0, the entry references another paging structure; if the bit is 1, the entry maps a page.

1. If HLAT paging is in use, a different mechanism is used to identify the first paging structure. See Section 4.5 for more information.

- If only 12 bits remain in the linear address, the current paging-structure entry always maps a page (bit 7 is used for other purposes).

If a paging-structure entry maps a page when more than 12 bits remain in the linear address, the entry identifies a page frame larger than 4 KBytes. For example, 32-bit paging uses the upper 10 bits of a linear address to locate the first paging-structure entry; 22 bits remain. If that entry maps a page, the page frame is 2^{22} Bytes = 4 MBytes. 32-bit paging can use 4-MByte pages if CR4.PSE = 1. The other paging modes can use 2-MByte pages (regardless of the value of CR4.PSE). 4-level paging and 5-level paging can use 1-GByte pages if the processor supports them (see Section 4.1.4).

Paging structures are given different names based on their uses in the translation process. Table 4-2 gives the names of the different paging structures. It also provides, for each structure, the source of the physical address used to locate it (CR3 or a different paging-structure entry); the bits in the linear address used to select an entry from the structure; and details of whether and how such an entry can map a page.

Table 4-2. Paging Structures in the Different Paging Modes

Paging Structure	Entry Name	Paging Mode	Physical Address of Structure	Bits Selecting Entry	Page Mapping
PML5 table	PML5E	32-bit, PAE, 4-level	N/A		
		5-level	CR3[1]	56:48	N/A (PS must be 0)
PML4 table	PML4E	32-bit, PAE	N/A		
		4-level	CR3[1]	47:39	N/A (PS must be 0)
		5-level	PML5E		
Page-directory-pointer table	PDPTE	32-bit	N/A		
		PAE	CR3	31:30	N/A (PS must be 0)
		4-level, 5-level	PML4E	38:30	1-GByte page if PS=1[2]
Page directory	PDE	32-bit	CR3	31:22	4-MByte page if PS=1[3]
		PAE, 4-level, 5-level	PDPTE	29:21	2-MByte page if PS=1
Page table	PTE	32-bit	PDE	21:12	4-KByte page
		PAE, 4-level, 5-level		20:12	

NOTES:

1. If HLAT paging is in use, a different mechanism is used to identify the first paging structure. See Section 4.5 for more information.
2. Not all processors support 1-GByte pages; see Section 4.1.4.
3. 32-bit paging ignores the PS flag in a PDE (and uses the entry to reference a page table) unless CR4.PSE = 1. Not all processors support 4-MByte pages with 32-bit paging; see Section 4.1.4.

4.3 32-BIT PAGING

A logical processor uses 32-bit paging if CR0.PG = 1 and CR4.PAE = 0. 32-bit paging translates 32-bit linear addresses to 40-bit physical addresses.[1] Although 40 bits corresponds to 1 TByte, linear addresses are limited to 32 bits; at most 4 GBytes of linear-address space may be accessed at any given time.

32-bit paging uses a hierarchy of paging structures to produce a translation for a linear address. CR3 is used to locate the first paging-structure, the page directory. Table 4-3 illustrates how CR3 is used with 32-bit paging.

32-bit paging may map linear addresses to either 4-KByte pages or 4-MByte pages. Figure 4-2 illustrates the translation process when it uses a 4-KByte page; Figure 4-3 covers the case of a 4-MByte page. The following items describe the 32-bit paging process in more detail as well has how the page size is determined:

- A 4-KByte naturally aligned page directory is located at the physical address specified in bits 31:12 of CR3 (see Table 4-3). A page directory comprises 1024 32-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
  - Bits 39:32 are all 0.
  - Bits 31:12 are from CR3.
  - Bits 11:2 are bits 31:22 of the linear address.
  - Bits 1:0 are 0.

Because a PDE is identified using bits 31:22 of the linear address, it controls access to a 4-Mbyte region of the linear-address space. Use of the PDE depends on CR4.PSE and the PDE's PS flag (bit 7):

- If CR4.PSE = 1 and the PDE's PS flag is 1, the PDE maps a 4-MByte page (see Table 4-4). The final physical address is computed as follows:
  - Bits 39:32 are bits 20:13 of the PDE.
  - Bits 31:22 are bits 31:22 of the PDE.[2]
  - Bits 21:0 are from the original linear address.
- If CR4.PSE = 0 or the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 31:12 of the PDE (see Table 4-5). A page table comprises 1024 32-bit entries (PTEs). A PTE is selected using the physical address defined as follows:
  - Bits 39:32 are all 0.
  - Bits 31:12 are from the PDE.
  - Bits 11:2 are bits 21:12 of the linear address.
  - Bits 1:0 are 0.
- Because a PTE is identified using bits 31:12 of the linear address, every PTE maps a 4-KByte page (see Table 4-6). The final physical address is computed as follows:
  - Bits 39:32 are all 0.
  - Bits 31:12 are from the PTE.
  - Bits 11:0 are from the original linear address.

If a paging-structure entry's P flag (bit 0) is 0 or if the entry sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. There is no translation for a linear address whose translation would use such a paging-structure entry; a reference to such a linear address causes a page-fault exception (see Section 4.7).

---

1. Bits in the range 39:32 are 0 in any physical address used by 32-bit paging except those used to map 4-MByte pages. If the processor does not support the PSE-36 mechanism, this is true also for physical addresses used to map 4-MByte pages. If the processor does support the PSE-36 mechanism and MAXPHYADDR < 40, bits in the range 39:MAXPHYADDR are 0 in any physical address used to map a 4-MByte page. (The corresponding bits are reserved in PDEs.) See Section 4.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.
2. The upper bits in the final physical address do not all come from corresponding positions in the PDE; the physical-address bits in the PDE are not all contiguous.

With 32-bit paging, there are reserved bits only if CR4.PSE = 1:

- If the P flag and the PS flag (bit 7) of a PDE are both 1, the bits reserved depend on MAXPHYADDR, and whether the PSE-36 mechanism is supported:[1]
  - If the PSE-36 mechanism is not supported, bits 21:13 are reserved.
  - If the PSE-36 mechanism is supported, bits 21:(M–19) are reserved, where M is the minimum of 40 and MAXPHYADDR.
- If the PAT is not supported:[2]
  - If the P flag of a PTE is 1, bit 7 is reserved.
  - If the P flag and the PS flag of a PDE are both 1, bit 12 is reserved.

(If CR4.PSE = 0, no bits are reserved with 32-bit paging.)

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 4.6.

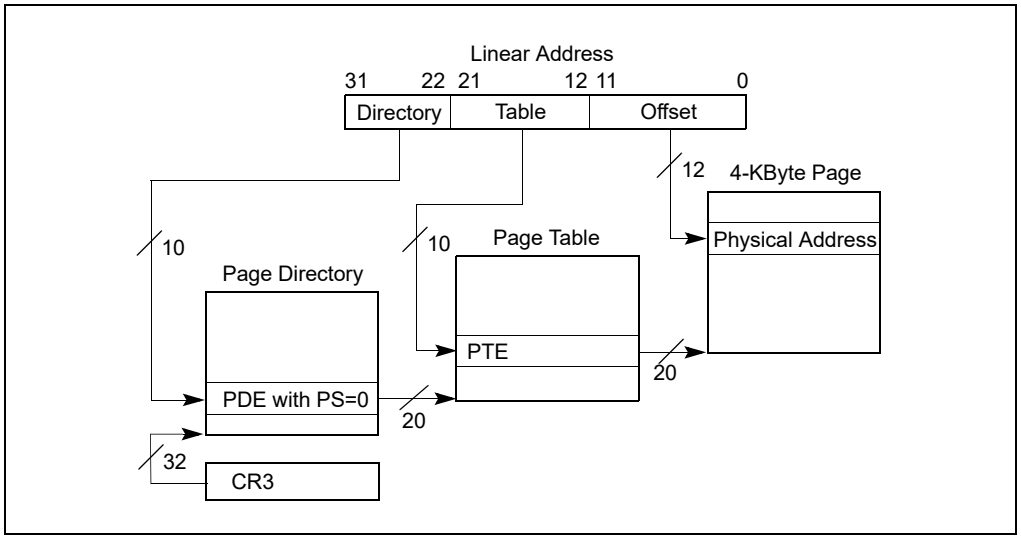


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

1. See Section 4.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.
2. See Section 4.1.4 for how to determine whether the PAT is supported.

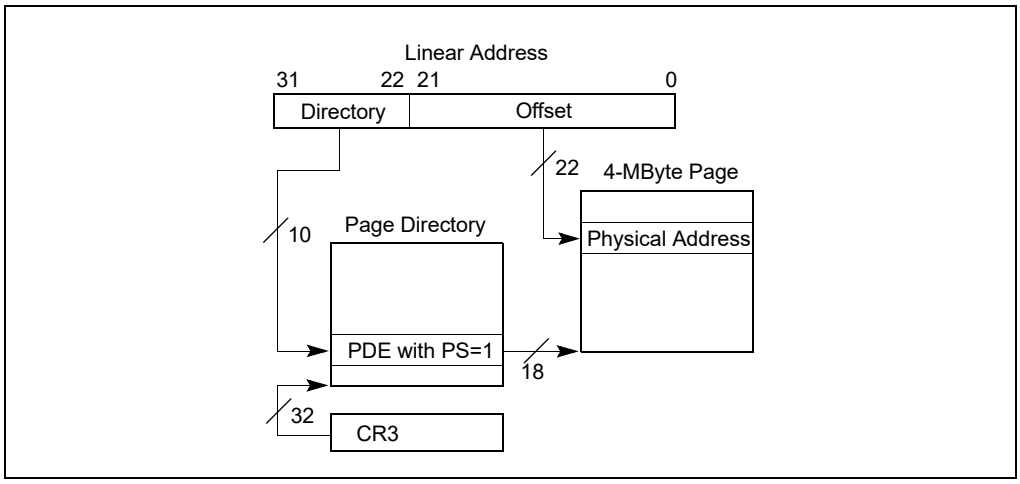


Figure 4-3. Linear-Address Translation to a 4-MByte Page using 32-Bit Paging

Figure 4-4 gives a summary of the formats of CR3 and the paging-structure entries with 32-bit paging. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are "not present"; bit 0 (P) and bit 7 (PS) are highlighted because they determine how such an entry is used.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page directory[1]																				Ignored							PCD	PWT	Ignored			**CR3**
Bits 31:22 of address of 4MB page frame											Reserved (must be 0)					Bits 39:32 of address[2]			PAT	Ignored			G	**1**	D	A	PCD	PWT	U/S	R/W	**1**	**PDE: 4MB page**
Address of page table																				Ignored				**0**	Ign	A	PCD	PWT	U/S	R/W	**1**	**PDE: page table**
Ignored																															**0**	**PDE: not present**
Address of 4KB page frame																				Ignored			G	PAT	D	A	PCD	PWT	U/S	R/W	**1**	**PTE: 4KB page**
Ignored																															**0**	**PTE: not present**

Figure 4-4. Formats of CR3 and Paging-Structure Entries with 32-Bit Paging

NOTES:

1. CR3 has 64 bits on processors supporting the Intel-64 architecture. These bits are ignored with 32-bit paging.
2. This example illustrates a processor in which MAXPHYADDR is 36. If this value is larger or smaller, the number of bits reserved in positions 20:13 of a PDE mapping a 4-MByte page will change.

Table 4-3. Use of CR3 with 32-Bit Paging

Bit Position(s)	Contents
2:0	Ignored
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory during linear-address translation (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory during linear-address translation (see Section 4.9)
11:5	Ignored
31:12	Physical address of the 4-KByte aligned page directory used for linear-address translation
63:32	Ignored (these bits exist only on processors supporting the Intel-64 architecture)

Table 4-4. Format of a 32-Bit Page-Directory Entry that Maps a 4-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-MByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 4-5)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0)[1]
(M–20):13	Bits (M–1):32 of physical address of the 4-MByte page referenced by this entry[2]
21:(M–19)	Reserved (must be 0)
31:22	Bits 31:22 of physical address of the 4-MByte page referenced by this entry

NOTES:

1. See Section 4.1.4 for how to determine whether the PAT is supported.
2. If the PSE-36 mechanism is not supported, M is 32, and this row does not apply. If the PSE-36 mechanism is supported, M is the minimum of 40 and MAXPHYADDR (this row does not apply if MAXPHYADDR = 32). See Section 4.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.

Table 4-5. Format of a 32-Bit Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-MByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	If CR4.PSE = 1, must be 0 (otherwise, this entry maps a 4-MByte page; see Table 4-4); otherwise, ignored
11:8	Ignored
31:12	Physical address of 4-KByte aligned page table referenced by this entry

Table 4-6. Format of a 32-Bit Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0)[1]
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
31:12	Physical address of the 4-KByte page referenced by this entry

NOTES:

1. See Section 4.1.4 for how to determine whether the PAT is supported.

4.4 PAE PAGING

A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0. PAE paging translates 32-bit linear addresses to 52-bit physical addresses.[1] Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 32 bits; at most 4 GBytes of linear-address space may be accessed at any given time.

With PAE paging, a logical processor maintains a set of four (4) PDPTE registers, which are loaded from an address in CR3. Linear address are translated using 4 hierarchies of in-memory paging structures, each located using one of the PDPTE registers. (This is different from the other paging modes, in which there is one hierarchy referenced by CR3.)

Section 4.4.1 discusses the PDPTE registers. Section 4.4.2 describes linear-address translation with PAE paging.

4.4.1 PDPTE Registers

When PAE paging is used, CR3 references the base of a 32-Byte **page-directory-pointer table**. Table 4-7 illustrates how CR3 is used with PAE paging.

Table 4-7. Use of CR3 with PAE Paging

Bit Position(s)	Contents
4:0	Ignored
31:5	Physical address of the 32-Byte aligned page-directory-pointer table used for linear-address translation
63:32	Ignored (these bits exist only on processors supporting the Intel-64 architecture)

The page-directory-pointer-table comprises four (4) 64-bit entries called PDPTEs. Each PDPTE controls access to a 1-GByte region of the linear-address space. Corresponding to the PDPTEs, the logical processor maintains a set of four (4) internal, non-architectural PDPTE registers, called PDPTE0, PDPTE1, PDPTE2, and PDPTE3. The logical processor loads these registers from the PDPTEs in memory as part of certain operations:

- If PAE paging would be in use following an execution of MOV to CR0 or MOV to CR4 (see Section 4.1.1) and the instruction is modifying any of CR0.CD, CR0.NW, CR0.PG, CR4.PAE, CR4.PGE, CR4.PSE, or CR4.SMEP; then the PDPTEs are loaded from the address in CR3.
- If MOV to CR3 is executed while the logical processor is using PAE paging, the PDPTEs are loaded from the address being loaded into CR3.
- If PAE paging is in use and a task switch changes the value of CR3, the PDPTEs are loaded from the address in the new CR3 value.
- Certain VMX transitions load the PDPTE registers. See Section 4.11.1.

Table 4-8 gives the format of a PDPTE. If any of the PDPTEs sets both the P flag (bit 0) and any reserved bit, the MOV to CR instruction causes a general-protection exception (#GP(0)) and the PDPTEs are not loaded.[2] As shown in Table 4-8, bits 2:1, 8:5, and 63:MAXPHYADDR are reserved in the PDPTEs.

1. If MAXPHYADDR < 52, bits in the range 51:MAXPHYADDR will be 0 in any physical address used by PAE paging. (The corresponding bits are reserved in the paging-structure entries.) See Section 4.1.4 for how to determine MAXPHYADDR.
2. On some processors, reserved bits are checked even in PDPTEs in which the P flag (bit 0) is 0.

Table 4-8. Format of a PAE Page-Directory-Pointer-Table Entry (PDPTE)

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page directory
2:1	Reserved (must be 0)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9)
8:5	Reserved (must be 0)
11:9	Ignored
(M–1):12	Physical address of 4-KByte aligned page directory referenced by this entry[1]
63:M	Reserved (must be 0)

NOTES:
1. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.

4.4.2 Linear-Address Translation with PAE Paging

PAE paging may map linear addresses to either 4-KByte pages or 2-MByte pages. Figure 4-5 illustrates the translation process when it produces a 4-KByte page; Figure 4-6 covers the case of a 2-MByte page. The following items describe the PAE paging process in more detail as well has how the page size is determined:

- Bits 31:30 of the linear address select a PDPTE register (see Section 4.4.1); this is PDPTE*i*, where *i* is the value of bits 31:30.[1] Because a PDPTE register is identified using bits 31:30 of the linear address, it controls access to a 1-GByte region of the linear-address space. If the P flag (bit 0) of PDPTE*i* is 0, the processor ignores bits 63:1, and there is no mapping for the 1-GByte region controlled by PDPTE*i*. A reference using a linear address in this region causes a page-fault exception (see Section 4.7).
- If the P flag of PDPTE*i* is 1, 4-KByte naturally aligned page directory is located at the physical address specified in bits 51:12 of PDPTE*i* (see Table 4-8 in Section 4.4.1). A page directory comprises 512 64-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
  - Bits 51:12 are from PDPTE*i*.
  - Bits 11:3 are bits 29:21 of the linear address.
  - Bits 2:0 are 0.

Because a PDE is identified using bits 31:21 of the linear address, it controls access to a 2-Mbyte region of the linear-address space. Use of the PDE depends on its PS flag (bit 7):

- If the PDE's PS flag is 1, the PDE maps a 2-MByte page (see Table 4-9). The final physical address is computed as follows:
  - Bits 51:21 are from the PDE.
  - Bits 20:0 are from the original linear address.
- If the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 51:12 of the PDE (see Table 4-10). A page table comprises 512 64-bit entries (PTEs). A PTE is selected using the physical address defined as follows:
  - Bits 51:12 are from the PDE.

1. With PAE paging, the processor does not use CR3 when translating a linear address (as it does in the other paging modes). It does not access the PDPTEs in the page-directory-pointer table during linear-address translation.

  - Bits 11:3 are bits 20:12 of the linear address.
  - Bits 2:0 are 0.
- Because a PTE is identified using bits 31:12 of the linear address, every PTE maps a 4-KByte page (see Table 4-11). The final physical address is computed as follows:
  - Bits 51:12 are from the PTE.
  - Bits 11:0 are from the original linear address.

If the P flag (bit 0) of a PDE or a PTE is 0 or if a PDE or a PTE sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. There is no translation for a linear address whose translation would use such a paging-structure entry; a reference to such a linear address causes a page-fault exception (see Section 4.7).

The following bits are reserved with PAE paging:

- If the P flag (bit 0) of a PDE or a PTE is 1, bits 62:MAXPHYADDR are reserved.
- If the P flag and the PS flag (bit 7) of a PDE are both 1, bits 20:13 are reserved.
- If IA32_EFER.NXE = 0 and the P flag of a PDE or a PTE is 1, the XD flag (bit 63) is reserved.
- If the PAT is not supported:[1]
  - If the P flag of a PTE is 1, bit 7 is reserved.
  - If the P flag and the PS flag of a PDE are both 1, bit 12 is reserved.

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 4.6.

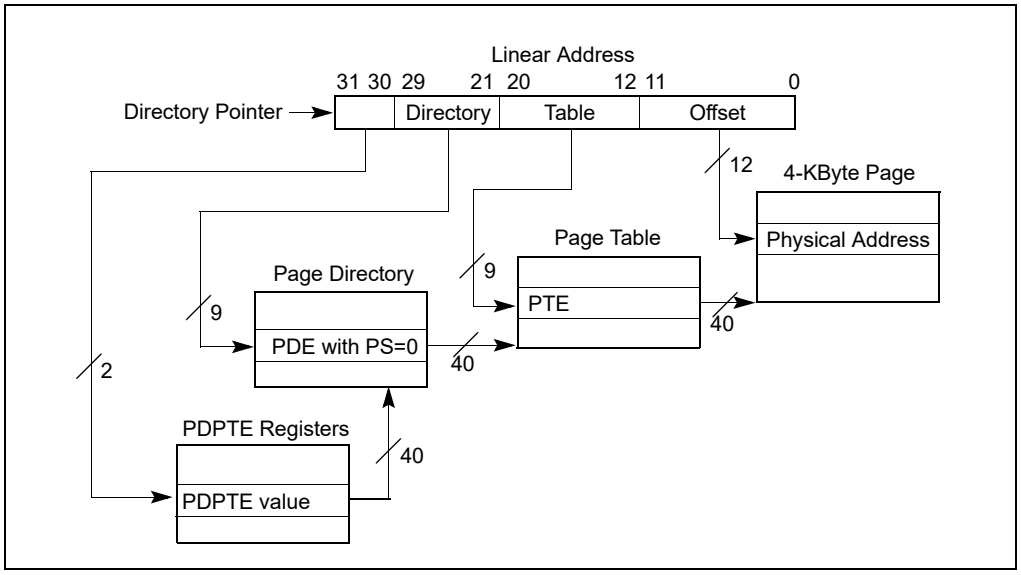


Figure 4-5. Linear-Address Translation to a 4-KByte Page using PAE Paging

1. See Section 4.1.4 for how to determine whether the PAT is supported.

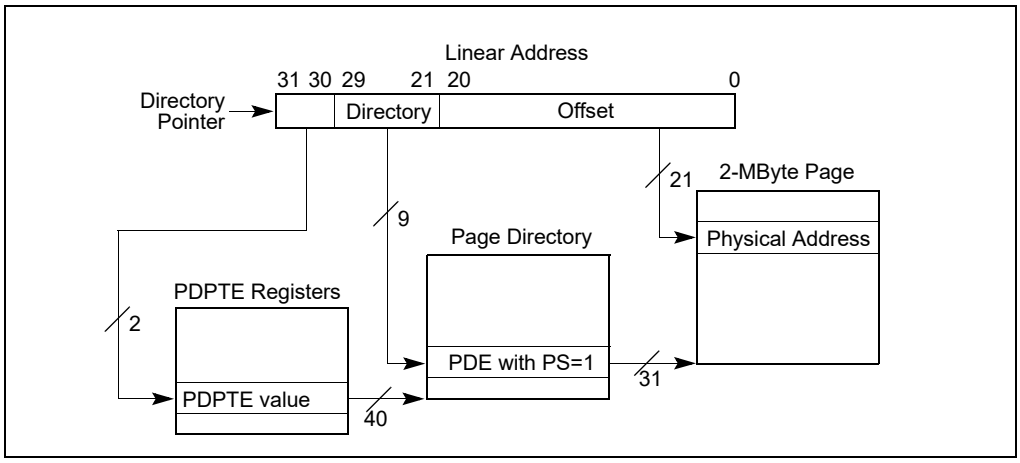


Figure 4-6. Linear-Address Translation to a 2-MByte Page using PAE Paging

Table 4-9. Format of a PAE Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 2-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 4-10)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0)[1]
20:13	Reserved (must be 0)
(M–1):21	Physical address of the 2-MByte page referenced by this entry
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:

1. See Section 4.1.4 for how to determine whether the PAT is supported.

Table 4-10. Format of a PAE Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 2-MByte page; see Table 4-9)
11:8	Ignored
(M–1):12	Physical address of 4-KByte aligned page table referenced by this entry
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-11. Format of a PAE Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0)[1]
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise

Table 4-11. Format of a PAE Page-Table Entry that Maps a 4-KByte Page (Contd.)

Bit Position(s)	Contents
11:9	Ignored
(M–1):12	Physical address of the 4-KByte page referenced by this entry
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:

1. See Section 4.1.4 for how to determine whether the PAT is supported.

Figure 4-7 gives a summary of the formats of CR3 and the paging-structure entries with PAE paging. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are "not present"; bit 0 (P) and bit 7 (PS) are highlighted because they determine how a paging-structure entry is used.

63	62–M	(M–1)–32	31–5	4	3	2	1	0	
Ignored[2]		Address of page-directory-pointer table		Ignored					**CR3**
Reserved[3]		Address of page directory		Ign.	Rsvd.	PCD	PWT	Rsvd	**1**	**PDPTE: present**
Ignored									**0**	**PDTPE: not present**
XD[4]	Reserved	Address of 2MB page frame	Reserved PAT Ign. G **1** D A PCD PWT U/S R/W						**1**	**PDE: 2MB page**
XD	Reserved	Address of page table	Ign. **0** Ign A PCD PWT U/S R/W						**1**	**PDE: page table**
Ignored									**0**	**PDE: not present**
XD	Reserved	Address of 4KB page frame	Ign. G PAT D A PCD PWT U/S R/W						**1**	**PTE: 4KB page**
Ignored									**0**	**PTE: not present**

Figure 4-7. Formats of CR3 and Paging-Structure Entries with PAE Paging

NOTES:

1. M is an abbreviation for MAXPHYADDR.
2. CR3 has 64 bits only on processors supporting the Intel-64 architecture. These bits are ignored with PAE paging.
3. Reserved fields must be 0.
4. If IA32_EFER.NXE = 0 and the P flag of a PDE or a PTE is 1, the XD flag (bit 63) is reserved.

4.5 4-LEVEL PAGING AND 5-LEVEL PAGING

Because the operation of 4-level paging and 5-level paging is very similar, they are described together in this section. The following items highlight the distinctions between the two paging modes:

- A logical processor uses 4-level paging if CR0.PG = 1, CR4.PAE = 1, IA32_EFER.LME = 1, and CR4.LA57 = 0. 4-level paging translates 48-bit linear addresses to 52-bit physical addresses.[1] Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 48 bits; at most 256 TBytes of linear-address space may be accessed at any given time.
- A logical processor uses 5-level paging if CR0.PG = 1, CR4.PAE = 1, IA32_EFER.LME = 1, and CR4.LA57 = 1. 5-level paging translates 57-bit linear addresses to 52-bit physical addresses. Thus, 5-level paging supports a linear-address space sufficient to access the entire physical-address space.

4.5.1 Ordinary Paging and HLAT Paging

There are two forms of 4-level paging and 5-level paging that differ principally with regard to how linear-address translation identifies the first paging structure.

The normal form is called **ordinary paging**, and it uses CR3 to locate the first paging structure, similar to what is done for 32-bit paging. Section 4.5.2 provides details of this use of CR3.

An alternative form of paging may be used with the VMX feature called hypervisor-managed linear-address translation (HLAT). Called **HLAT paging**, this form is used only in VMX non-root operation and only if the "enable HLAT" VM-execution control is 1.[2] HLAT paging locates the first paging structure using a VM-execution control field in the VMCS called the **HLAT pointer** (**HLATP**). Section 4.5.3 provides details.

Whether HLAT paging is used to translate a specific linear address depends on the address and on the value of a VM-execution control field in the VMCS called the **HLAT prefix size**:

- If the HLAT prefix size is zero, every linear address is translated using HLAT paging.
- If the HLAT prefix size is not zero, a linear address is translated using HLAT paging if bit 63 of the address is 1.[3] The address is translated using ordinary paging if bit 63 of the address is 0.

In some cases, HLAT paging may specify that a translation of a linear address must be restarted. When this occurs, the linear address is then translated using ordinary paging (starting with a paging structure identified using CR3). The situations leading to this restart are detailed in Section 4.5.4, and additional details of the restart process are given in Section 4.5.5.

4.5.2 Use of CR3 with Ordinary 4-Level Paging and 5-Level Paging

Ordinary 4-level paging and 5-level paging each translate linear addresses using a hierarchy of in-memory paging structures located using the contents of CR3, which is used to locate the first paging structure. For 4-level paging, this is the PML4 table, and for 5-level paging it is the PML5 table. Use of CR3 with 4-level paging and 5-level paging depends on whether process-context identifiers (PCIDs) have been enabled by setting CR4.PCIDE:

- Table 4-12 illustrates how CR3 is used with 4-level paging and 5-level paging if CR4.PCIDE = 0.

Table 4-12. Use of CR3 with 4-Level Paging and 5-level Paging and CR4.PCIDE = 0

Bit Position(s)	Contents
2:0	Ignored

1. If MAXPHYADDR < 52, bits in the range 51:MAXPHYADDR will be 0 in any physical address used by 4-level paging. (The corresponding bits are reserved in the paging-structure entries.) See Section 4.1.4 for how to determine MAXPHYADDR.
2. HLAT paging is used only with 4-level paging and 5-level paging. It is never used with 32-bit paging or PAE paging, regardless of the value of the "enable HLAT" VM-execution control.
3. This behavior applies if the CPU enumerates a maximum HLAT prefix size of 1 in IA32_VMX_EPT_VPID_CAP[53:48] (see Appendix A.10). Behavior when a different value is enumerated is not currently defined.

Table 4-12. Use of CR3 with 4-Level Paging and 5-level Paging and CR4.PCIDE = 0 (Contd.)

Bit Position(s)	Contents
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the PML4 table or PML5 table during linear-address translation (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the PML4 table or PML5 table during linear-address translation (see Section 4.9.2)
11:5	Ignored
M–1:12	Physical address of the 4-KByte aligned PML4 table or PML5 table used for linear-address translation[1]
63:M	Reserved (must be 0)

NOTES:

1. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.

- Table 4-13 illustrates how CR3 is used with 4-level paging and 5-level paging if CR4.PCIDE = 1.

Table 4-13. Use of CR3 with 4-Level Paging and 5-Level Paging and CR4.PCIDE = 1

Bit Position(s)	Contents
11:0	PCID (see Section 4.10.1)[1]
M–1:12	Physical address of the 4-KByte aligned PML4 table used for linear-address translation[2]
63:M	Reserved (must be 0)[3]

NOTES:

1. Section 4.9.2 explains how the processor determines the memory type used to access the PML4 table during linear-address translation with CR4.PCIDE = 1.
2. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.
3. See Section 4.10.4.1 for use of bit 63 of the source operand of the MOV to CR3 instruction.

After software modifies the value of CR4.PCIDE, the logical processor immediately begins using CR3 as specified for the new value. For example, if software changes CR4.PCIDE from 1 to 0, the current PCID immediately changes from CR3[11:0] to 000H (see also Section 4.10.4.1). In addition, the logical processor subsequently determines the memory type used to access the PML4 table using CR3.PWT and CR3.PCD, which had been bits 4:3 of the PCID.

4.5.3 Use of HLATP with HLAT 4-Level Paging and 5-Level Paging

With HLAT paging, 4-level paging and 5-level paging each translate linear addresses using a hierarchy of in-memory paging structures located using the value of HLATP (a VM-execution control field in the VMCS), which is used to locate the first paging structure. For 4-level paging, this is the PML4 table, and for 5-level paging it is the PML5 table.

HLATP has the same format as that given for CR3 in Table 4-12, with the exception that bits 2:0 and bits 11:5 are reserved and must be zero (these bits are checked by VM entry). HLATP does not contain a PCID value. HLAT paging with CR4.PCIDE = 1 uses the PCID value in CR3[11:0].

4.5.4 Linear-Address Translation with 4-Level Paging and 5-Level Paging

4-level paging and 5-level paging may map linear addresses to 4-KByte pages, 2-MByte pages, or 1-GByte pages.[1] Figure 4-8 illustrates the translation process for 4-level paging when it produces a 4-KByte page; Figure 4-9 covers the case of a 2-MByte page, and Figure 4-10 the case of a 1-GByte page. (The process for 5-level paging is similar.)

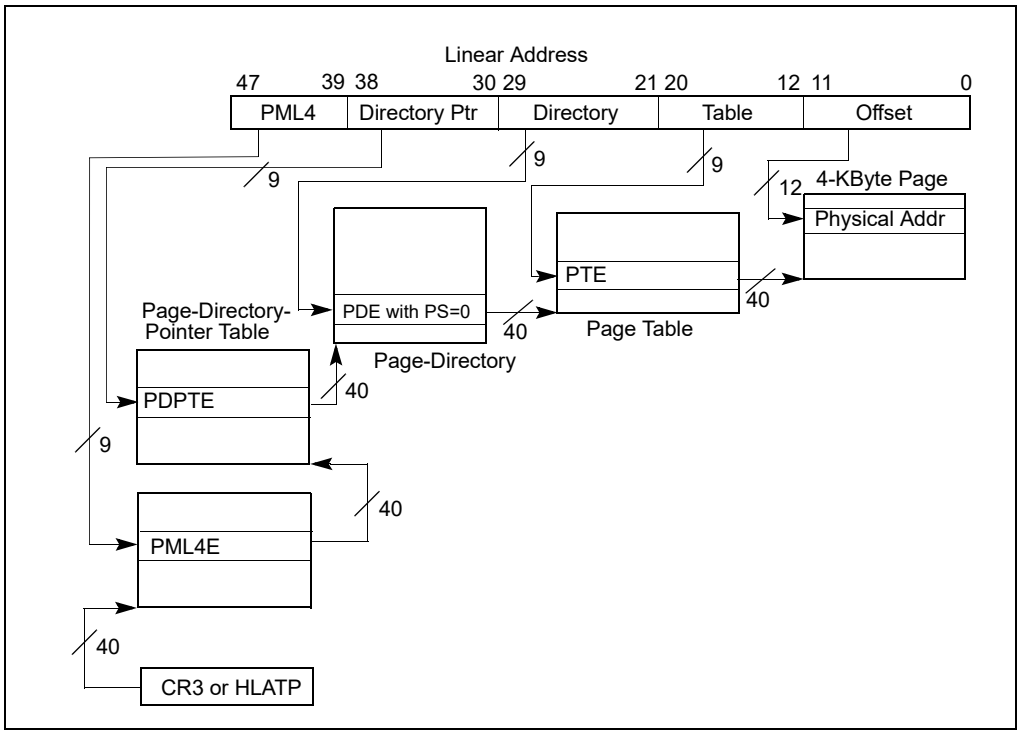


Figure 4-8. Linear-Address Translation to a 4-KByte Page using 4-Level Paging

1. Not all processors support 1-GByte pages; see Section 4.1.4.

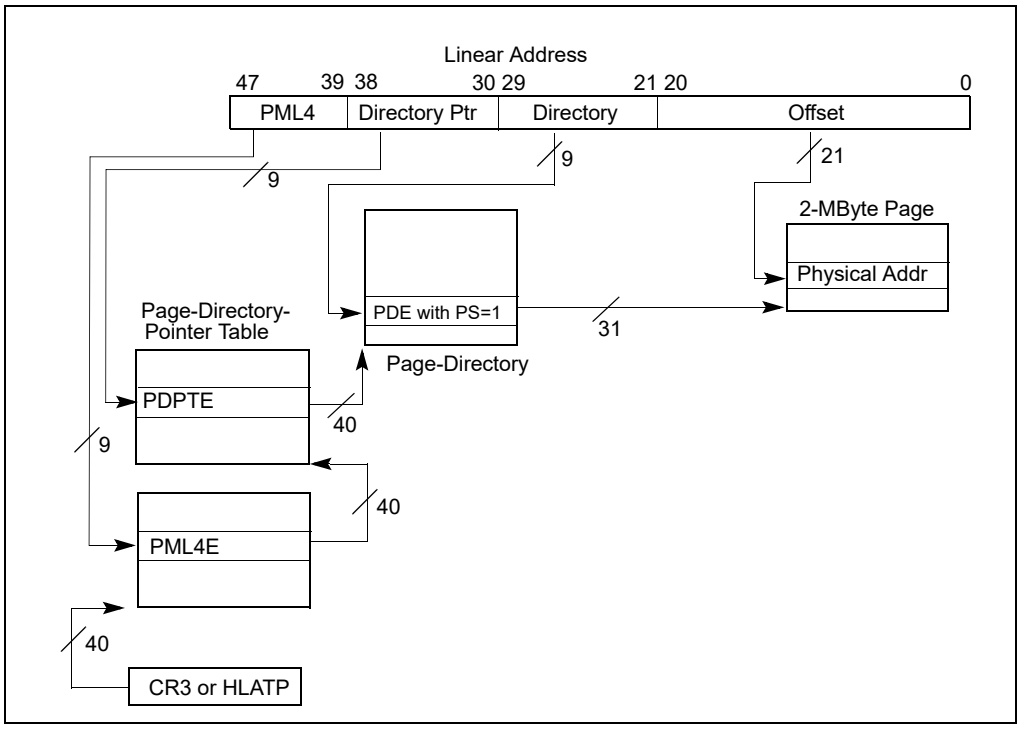


Figure 4-9. Linear-Address Translation to a 2-MByte Page using 4-Level Paging

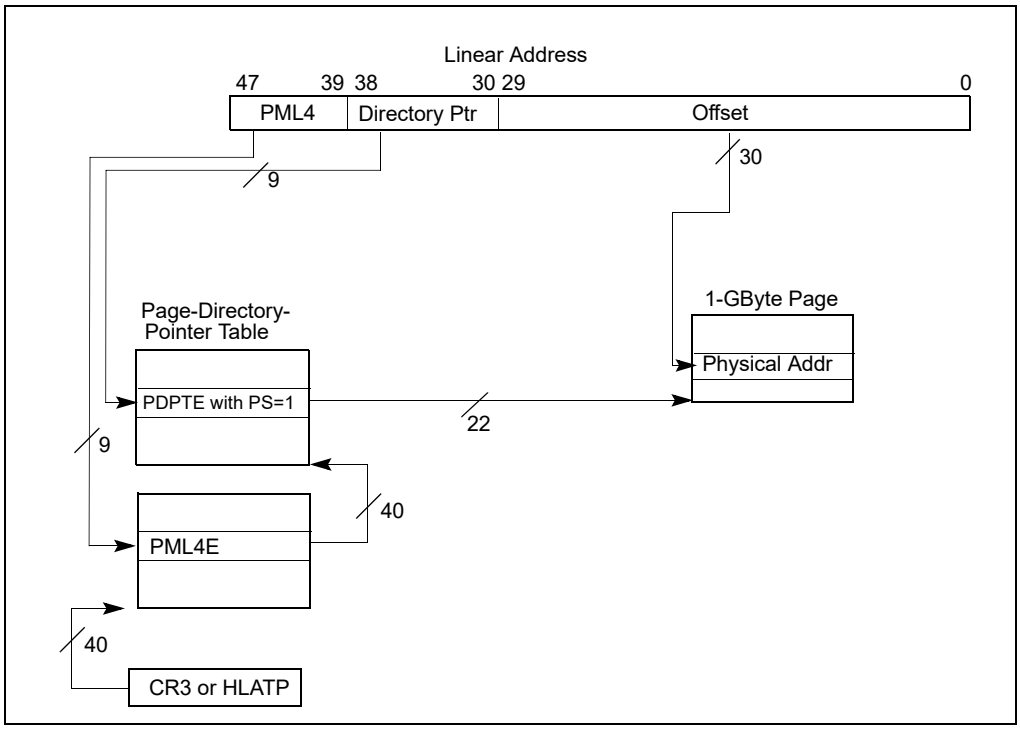


Figure 4-10. Linear-Address Translation to a 1-GByte Page using 4-Level Paging

4-level paging and 5-level paging associate with each linear address a **protection key**. Section 4.6 explains how the processor uses the protection key in its determination of the access rights of each linear address.

The remainder of this section describes the translation process used by 4-level paging and 5-level paging in more detail, as well has how the page size and protection key are determined. Because the process used by the two paging modes is similar, they are described together, with any differences identified, in the following items:

- With 5-level paging, a 4-KByte naturally aligned PML5 table is located at the physical address specified in bits 51:12 of CR3 (see Table 4-12). (4-level paging does not use a PML5 table and omits this step.) A PML5 table comprises 512 64-bit entries (PML5Es). A PML5E is selected using the physical address defined as follows:
  - Bits 51:12 are from CR3 or HLATP.
  - Bits 11:3 are bits 56:48 of the linear address.
  - Bits 2:0 are all 0.

  Because a PML5E is identified using bits 56:48 of the linear address, it controls access to a 256-TByte region of the linear-address space.

  With HLAT paging, if bit 11 of the PML5E is 1, translation is restarted with ordinary paging with a maximum page size of 256-TBytes (see Section 4.5.5). Otherwise, the translation process continues as described in the next item.
- A 4-KByte naturally aligned PML4 table is located at the physical address specified in bits 51:12 of CR3 (for 4-level paging; see Table 4-12) or in bits 51:12 of the PML4E (for 5-level paging; see Table 4-14). A PML4 table comprises 512 64-bit entries (PML4Es). A PML4E is selected using the physical address defined as follows:
  - Bits 51:12 are from CR3 or the HLATP (for 4-level paging) or in bits 51:12 of the PML4E (for 5-level paging).
  - Bits 11:3 are bits 47:39 of the linear address.
  - Bits 2:0 are all 0.

  Because a PML4E is identified using bits 47:39 of the linear address, it controls access to a 512-GByte region of the linear-address space.

  With HLAT paging, if bit 11 of the PML4E is 1, translation is restarted with ordinary paging with a maximum page size of 512-GBytes (see Section 4.5.5). Otherwise, the translation process continues as described in the next item.
- A 4-KByte naturally aligned page-directory-pointer table is located at the physical address specified in bits 51:12 of the PML4E (see Table 4-15). A page-directory-pointer table comprises 512 64-bit entries (PDPTEs). A PDPTE is selected using the physical address defined as follows:
  - Bits 51:12 are from the PML4E.
  - Bits 11:3 are bits 38:30 of the linear address.
  - Bits 2:0 are all 0.

Because a PDPTE is identified using bits 47:30 of the linear address, it controls access to a 1-GByte region of the linear-address space.

With HLAT paging, if bit 11 of the PDPTE is 1, translation is restarted with ordinary paging with a maximum page size of 1-GByte (see Section 4.5.5). Otherwise, the translation process continues as described below.

Use of the PDPTE depends on its PS flag (bit 7):[1]

- If the PDPTE's PS flag is 1, the PDPTE maps a 1-GByte page (see Table 4-16). The final physical address is computed as follows:
  - Bits 51:30 are from the PDPTE.
  - Bits 29:0 are from the original linear address.

  The linear address's protection key is the value of bits 62:59 of the PDPTE (see Section 4.6.2).

1. The PS flag of a PDPTE is reserved and must be 0 (if the P flag is 1) if 1-GByte pages are not supported. See Section 4.1.4 for how to determine whether 1-GByte pages are supported.

- If the PDPTE's PS flag is 0, a 4-KByte naturally aligned page directory is located at the physical address specified in bits 51:12 of the PDPTE (see Table 4-17). A page directory comprises 512 64-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
  - Bits 51:12 are from the PDPTE.
  - Bits 11:3 are bits 29:21 of the linear address.
  - Bits 2:0 are all 0.

Because a PDE is identified using bits 47:21 of the linear address, it controls access to a 2-MByte region of the linear-address space.

With HLAT paging, if bit 11 of the PDE is 1, translation is restarted with ordinary paging with a maximum page size of 2-MBytes (see Section 4.5.5). Otherwise, the translation process continues as described below.

Use of the PDE depends on its PS flag:

- If the PDE's PS flag is 1, the PDE maps a 2-MByte page (see Table 4-18). The final physical address is computed as follows:
  - Bits 51:21 are from the PDE.
  - Bits 20:0 are from the original linear address.

  The linear address's protection key is the value of bits 62:59 of the PDE (see Section 4.6.2).
- If the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 51:12 of the PDE (see Table 4-19). A page table comprises 512 64-bit entries (PTEs). A PTE is selected using the physical address defined as follows:
  - Bits 51:12 are from the PDE.
  - Bits 11:3 are bits 20:12 of the linear address.
  - Bits 2:0 are all 0.
- Because a PTE is identified using bits 47:12 of the linear address, every PTE maps a 4-KByte page (see Table 4-20).

  With HLAT paging, if bit 11 of the PTE is 1, translation is restarted with ordinary paging with a maximum page size of 4-KBytes (see Section 4.5.5). Otherwise, the final physical address is computed as follows:
  - Bits 51:12 are from the PTE.
  - Bits 11:0 are from the original linear address.

  The linear address's protection key is the value of bits 62:59 of the PTE (see Section 4.6.2).

If a paging-structure entry's P flag (bit 0) is 0 or if the entry sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. There is no translation for a linear address whose translation would use such a paging-structure entry; a reference to such a linear address causes a page-fault exception (see Section 4.7).

The following bits in a paging-structure entry are reserved with 4-level paging and 5-level paging (assuming that the entry's P flag is 1):

- Bits 51:MAXPHYADDR are reserved in every paging-structure entry.
- The PS flag is reserved in a PML5E or a PML4E.
- If 1-GByte pages are not supported, the PS flag is reserved in a PDPTE.[1]
- If the PS flag in a PDPTE is 1, bits 29:13 of the entry are reserved.
- If the PS flag in a PDE is 1, bits 20:13 of the entry are reserved.
- If IA32_EFER.NXE = 0, the XD flag (bit 63) is reserved in every paging-structure entry.

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 4.6.

1. See Section 4.1.4 for how to determine whether 1-GByte pages are supported.

Figure 4-11 gives a summary of the formats of CR3 and the 4-level and 5-level paging-structure entries. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are "not present"; bit 0 (P) and bit 7 (PS) are highlighted because they determine how a paging-structure entry is used.

Table 4-14. Format of a PML5 Entry (PML5E) that References a PML4 Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a PML4 table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 256-TByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 256-TByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the PML4 table referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the PML4 table referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Reserved (must be 0)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
M–1:12	Physical address of 4-KByte aligned PML4 table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 256-TByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-15. Format of a PML4 Entry (PML4E) that References a Page-Directory-Pointer Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page-directory-pointer table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 512-GByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 512-GByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored

Table 4-15. Format of a PML4 Entry (PML4E) that References a Page-Directory-Pointer Table (Contd.)

Bit Position(s)	Contents
7 (PS)	Reserved (must be 0)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
M–1:12	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 512-GByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-16. Format of a Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 1-GByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 1-GByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 1-GByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 1-GByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 1-GByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page directory; see Table 4-17)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
10:9	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
12 (PAT)	Indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2)[1]
29:13	Reserved (must be 0)
(M–1):30	Physical address of the 1-GByte page referenced by this entry

Table 4-16. Format of a Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GByte Page (Contd.)

Bit Position(s)	Contents
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1 or CR4.PKS = 1, this may control the page's access rights (see Section 4.6.2); otherwise, it is ignored and not used to control access rights.
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 1-GByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:

1. The PAT is supported on all processors that support 4-level paging.

Table 4-17. Format of a Page-Directory-Pointer-Table Entry (PDPTE) that References a Page Directory

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page directory
1 (R/W)	Read/write; if 0, writes may not be allowed to the 1-GByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 1-GByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 1-GByte page; see Table 4-16)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
(M–1):12	Physical address of 4-KByte aligned page directory referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 1-GByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-18. Format of a Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 2-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 4-19)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
10:9	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
12 (PAT)	Indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
20:13	Reserved (must be 0)
(M–1):21	Physical address of the 2-MByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1 or CR4.PKS = 1, this may control the page's access rights (see Section 4.6.2); otherwise, it is ignored and not used to control access rights.
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-19. Format of a Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 2-MByte page; see Table 4-18)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
(M–1):12	Physical address of 4-KByte aligned page table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Table 4-20. Format of a Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
10:9	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
(M–1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1 or CR4.PKS = 1, this may control the page's access rights (see Section 4.6.2); otherwise, it is ignored and not used to control access rights.
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

63	62	61	60	59	58	57	56	55	54	53	52	51	M[1]	M-1	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved[2]														Address of PML4 table (4-level paging) or PML5 table (5-level paging)																								Ignored								PCD	PWT	Ign.			CR3
XD[3]	Ignored												Rsvd.	Address of PML4 table																							R[4]	Ign.							Rsvd	Ign	A	PCD	PWT	U/S	R/W	1	PML5E: present
Ignored																																																	0	PML5E: not present
XD	Ignored												Rsvd.	Address of page-directory-pointer table																							R	Ign.							Rsvd	Ign	A	PCD	PWT	U/S	R/W	1	PML4E: present
Ignored																																																	0	PML4E: not present
XD	Prot. Key[5]		Ignored										Rsvd.	Address of 1GB page frame											Reserved												PAT	R	Ign.		G	1	D	A	PCD	PWT	U/S	R/W	1	PDPTE: 1GB page
XD	Ignored												Rsvd.	Address of page directory																							R	Ign.				0	Ign	A	PCD	PWT	U/S	R/W	1	PDPTE: page directory
Ignored																																																	0	PDTPE: not present
XD	Prot. Key		Ignored										Rsvd.	Address of 2MB page frame																							Reserved								PAT	R	Ign.		G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 2MB page
XD	Ignored												Rsvd.	Address of page table																							R	Ign.				0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
Ignored																																																	0	PDE: not present
XD	Prot. Key		Ignored										Rsvd.	Address of 4KB page frame																							R	Ign.		G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored																																																	0	PTE: not present

Figure 4-11. Formats of CR3 and Paging-Structure Entries with 4-Level Paging and 5-Level Paging

NOTES:

1. M is an abbreviation for MAXPHYADDR.
2. Reserved fields must be 0.
3. If IA32_EFER.NXE = 0 and the P flag of a paging-structure entry is 1, the XD flag (bit 63) is reserved.
4. Bit 11 is R (restart) only for HLAT paging; it is ignored for ordinary paging.
5. The protection key is used only if software has enabled the appropriate feature; see Section 4.6.2. It is ignored otherwise.

4.5.5 Restart of HLAT Paging

As noted in Section 4.5.1, HLAT paging may specify that a translation of a linear address must be restarted. Specifically, this occurs when HLAT paging encounters a paging-structure entry that sets bit 11 (see Section 4.5.4).

When this occurs, translation of the linear address is restarted using ordinary paging (starting with a paging structure identified using CR3). The restarted translation proceeds just as if the HLAT feature were not enabled. The entire linear address is translated again, including those portions that had been used by HLAT paging prior to the restart.

The process of restarting HLAT paging (using ordinary paging) always specifies a maximum page size to be used when a resulting translation is cached in the TLBs. This maximum page size depends on the level of the paging-structure entry that restarts the translation by setting bit 11; details are given in Section 4.5.4. The page size of the translation produced by the restarted process is never greater than this maximum page size. See Section 4.10.2.2 for more discussion.

4.6 ACCESS RIGHTS

There is a translation for a linear address if the processes described in Section 4.3, Section 4.4.2, and Section 4.5 (depending upon the paging mode) completes and produces a physical address. Whether an access is permitted by a translation is determined by the access rights specified by the paging-structure entries controlling the translation;[1] paging-mode modifiers in CR0, CR4, and the IA32_EFER MSR; EFLAGS.AC; and the mode of the access.

Section 4.6.1 describes how the processor determines the access rights for each linear address. Section 4.6.2 provides additional information about how protection keys contribute to access-rights determination. (They do so only with 4-level paging and 5-level paging, and only if CR4.PKE = 1 or CR4.PKS = 1.)

NOTE

If HLAT paging is restarted, permissions are determined only by the access rights specified by the paging-structure entries that the subsequent ordinary paging used to translate the linear address. The access rights specified by the entries used earlier by HLAT paging do not apply.

4.6.1 Determination of Access Rights

Every access to a linear address is either a **supervisor-mode access** or a **user-mode access**. For all instruction fetches and most data accesses, this distinction is determined by the current privilege level (CPL): accesses made while CPL < 3 are supervisor-mode accesses, while accesses made while CPL = 3 are user-mode accesses.

Some operations implicitly access system data structures with linear addresses; the resulting accesses to those data structures are supervisor-mode accesses regardless of CPL. Examples of such accesses include the following: accesses to the global descriptor table (GDT) or local descriptor table (LDT) to load a segment descriptor; accesses to the interrupt descriptor table (IDT) when delivering an interrupt or exception; and accesses to the task-state segment (TSS) as part of a task switch or change of CPL. All these accesses are called **implicit supervisor-mode accesses** regardless of CPL. Other accesses made while CPL < 3 are called **explicit supervisor-mode accesses**.

Access rights are also controlled by the **mode** of a linear address as specified by the paging-structure entries controlling the translation of the linear address. If the U/S flag (bit 2) is 0 in at least one of the paging-structure entries, the address is a **supervisor-mode address**. Otherwise, the address is a **user-mode address**.

When the shadow-stack feature of control-flow enforcement technology (CET) is enabled, certain accesses to linear addresses are considered **shadow-stack accesses** (see Section 18.2, "Shadow Stacks"in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*). Like ordinary data accesses, each shadow-stack access is defined as being either a user access or a supervisor access. In general, a shadow-stack access is a user access if CPL = 3 and a supervisor access if CPL < 3. The WRUSS instruction is an exception; although it can be executed only if CPL = 0, the processor treats its shadow-stack accesses as user accesses.

1. With PAE paging, the PDPTEs do not determine access rights.

Shadow-stack accesses are allowed only to **shadow-stack addresses**. A linear address is a shadow-stack address if the following are true of the translation of the linear address: (1) the R/W flag (bit 1) is 0 and the dirty flag (bit 6) is 1 in the paging-structure entry that maps the page containing the linear address; and (2) the R/W flag is 1 in every other paging-structure entry controlling the translation of the linear address.

The following items detail how paging determines access rights (only the items noted explicitly apply to shadow-stack accesses):

- For supervisor-mode accesses:
  - Data may be read (implicitly or explicitly) from any supervisor-mode address with a protection key for which read access is permitted (see Section 4.6.2).
  - Data reads from user-mode pages.
    Access rights depend on the value of CR4.SMAP:
    - If CR4.SMAP = 0, data may be read from any user-mode address with a protection key for which read access is permitted (see Section 4.6.2).
    - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
      - If EFLAGS.AC = 1 and the access is explicit, data may be read from any user-mode address with a protection key for which read access is permitted (see Section 4.6.2).
      - If EFLAGS.AC = 0 or the access is implicit, data may not be read from any user-mode address.
  - Data writes to supervisor-mode addresses.
    Access rights depend on the value of CR0.WP:
    - If CR0.WP = 0, data may be written to any supervisor-mode address with a protection key for which write access is permitted (see Section 4.6.2).
    - If CR0.WP = 1, data may be written to any supervisor-mode address with a translation for which the R/W flag (bit 1) is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted (see Section 4.6.2); data may not be written to any supervisor-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
  - Data writes to user-mode addresses.
    Access rights depend on the value of CR0.WP:
    - If CR0.WP = 0, access rights depend on the value of CR4.SMAP:
      - If CR4.SMAP = 0, data may be written to any user-mode address with a protection key for which write access is permitted (see Section 4.6.2).
      - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
        - If EFLAGS.AC = 1 and the access is explicit, data may be written to any user-mode address with a protection key for which write access is permitted (see Section 4.6.2).
        - If EFLAGS.AC = 0 or the access is implicit, data may not be written to any user-mode address.
    - If CR0.WP = 1, access rights depend on the value of CR4.SMAP:
      - If CR4.SMAP = 0, data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted (see Section 4.6.2); data may not be written to any user-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
      - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
        - If EFLAGS.AC = 1 and the access is explicit, data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted (see Section 4.6.2);

data may not be written to any user-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.

- If EFLAGS.AC = 0 or the access is implicit, data may not be written to any user-mode address.

— Instruction fetches from supervisor-mode addresses.

- For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any supervisor-mode address.
- For other paging modes with IA32_EFER.NXE = 1, instructions may be fetched from any supervisor-mode address with a translation for which the XD flag (bit 63) is 0 in every paging-structure entry controlling the translation; instructions may not be fetched from any supervisor-mode address with a translation for which the XD flag is 1 in any paging-structure entry controlling the translation.

— Instruction fetches from user-mode addresses.
Access rights depend on the values of CR4.SMEP:

- If CR4.SMEP = 0, access rights depend on the paging mode and the value of IA32_EFER.NXE:
  — For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any user-mode address.
  — For other paging modes with IA32_EFER.NXE = 1, instructions may be fetched from any user-mode address with a translation for which the XD flag is 0 in every paging-structure entry controlling the translation; instructions may not be fetched from any user-mode address with a translation for which the XD flag is 1 in any paging-structure entry controlling the translation.
- If CR4.SMEP = 1, instructions may not be fetched from any user-mode address.

— Supervisor-mode shadow-stack accesses are allowed only to supervisor-mode shadow-stack addresses (see above).

- For user-mode accesses:

— Data reads.
Access rights depend on the mode of the linear address:

- Data may be read from any user-mode address with a protection key for which read access is permitted (see Section 4.6.2).
- Data may not be read from any supervisor-mode address.

— Data writes.
Access rights depend on the mode of the linear address:

- Data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted (see Section 4.6.2).
- Data may not be written to any supervisor-mode address.

— Instruction fetches.
Access rights depend on the mode of the linear address, the paging mode, and the value of IA32_EFER.NXE:

- For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any user-mode address.
- For other paging modes with IA32_EFER.NXE = 1, instructions may be fetched from any user-mode address with a translation for which the XD flag is 0 in every paging-structure entry controlling the translation.
- Instructions may not be fetched from any supervisor-mode address.

— User-mode shadow-stack accesses made outside enclave mode are allowed only to user-mode shadow-stack addresses (see above). User-mode shadow-stack accesses made in enclave mode are treated like ordinary data accesses (see above).

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). These structures may include information about access rights. The processor may enforce access rights based on the TLBs and paging-structure caches instead of on the paging structures in memory.

This fact implies that, if software modifies a paging-structure entry to change access rights, the processor might not use that change for a subsequent access to an affected linear address (see Section 4.10.4.3). See Section 4.10.4.2 for how software can ensure that the processor uses the modified access rights.

4.6.2 Protection Keys

4-level paging and 5-level paging associate a 4-bit protection key with each linear address (the protection key located in bits 62:59 of the paging-structure entry that mapped the page containing the linear address; see Section 4.5). Two protection key features control accesses to linear addresses based on their protection keys:

- If CR4.PKE = 1, the PKRU register determines, for each protection key, whether user-mode addresses with that protection key may be read or written.
- If CR4.PKS = 1, the IA32_PKRS MSR (MSR index 6E1H) determines, for each protection key, whether supervisor-mode addresses with that protection key may be read or written.

32-bit paging and PAE paging do not associate linear addresses with protection keys. For the purposes of Section 4.6.1, reads and writes are implicitly permitted for all protection keys with either of those paging modes.

The PKRU register (protection-key rights for user pages) is a 32-bit register with the following format: for each *i* ($0 \le i \le 15$), PKRU[2*i*] is the **access-disable bit** for protection key *i* (AD*i*); PKRU[2*i*+1] is the **write-disable bit** for protection key *i* (WD*i*). The IA32_PKRS MSR has the same format (bits 63:32 of the MSR are reserved and must be zero).

Software can use the RDPKRU and WRPKRU instructions with ECX = 0 to read and write PKRU. In addition, the PKRU register is XSAVE-managed state and can thus be read and written by instructions in the XSAVE feature set. See Chapter 13, "Managing State Using the XSAVE Feature Set," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* for more information about the XSAVE feature set.

Software can use the RDMSR and WRMSR instructions to read and write the IA32_PKRS MSR. Writes to the IA32_PKRS MSR using WRMSR are not serializing. The IA32_PKRS MSR is not XSAVE-managed.

How a linear address's protection key controls access to the address depends on the mode of the linear address:

- A linear address's protection key controls only data accesses to the address. It does not in any way affect instructions fetches from the address.
- If CR4.PKE = 0, the protection key of a user-mode address does not control data accesses to the address (for the purposes of Section 4.6.1, reads and writes of user-mode addresses are implicitly permitted for all protection keys).

  If CR4.PKE = 1, use of the protection key *i* of a user-mode address depends on the value of the PKRU register:

  — If AD*i* = 1, no data accesses are permitted.

  — If WD*i* = 1, permission may be denied to certain data write accesses:

    - User-mode write accesses are not permitted.
    - Supervisor-mode write accesses are not permitted if CR0.WP = 1. (If CR0.WP = 0, WD*i* does not affect supervisor-mode write accesses to user-mode addresses with protection key *i*.)
- If CR4.PKS = 0, the protection key of a supervisor-mode address does not control data accesses to the address (for the purposes of Section 4.6.1, reads and writes of supervisor-mode addresses are implicitly permitted for all protection keys).

  If CR4.PKS = 1, use of the protection key *i* of a supervisor-mode address depends on the value of the IA32_PKRS MSR:

  — If AD*i* = 1, no data accesses are permitted.

  — If WD*i* = 1, write accesses are not permitted if CR0.WP = 1. (If CR0.WP = 0, IA32_PKRS.WD*i* does not affect write accesses to supervisor-mode addresses with protection key *i*.)

Protection keys apply to shadow-stack accesses just as they do to ordinary data accesses.

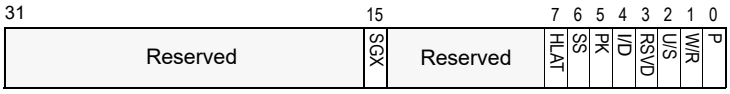
4.7 PAGE-FAULT EXCEPTIONS

Accesses using linear addresses may cause **page-fault exceptions** (#PF; exception 14). An access to a linear address may cause a page-fault exception for either of two reasons: (1) there is no translation for the linear address; or (2) there is a translation for the linear address, but its access rights do not permit the access.

As noted in Section 4.3, Section 4.4.2, and Section 4.5, there is no translation for a linear address if the translation process for that address would use a paging-structure entry in which the P flag (bit 0) is 0 or one that sets a reserved bit.[1] If there is a translation for a linear address, its access rights are determined as specified in Section 4.6.

When Intel® Software Guard Extensions (Intel® SGX) are enabled, the processor may deliver exception 14 for reasons unrelated to paging. See Section 34.3, "Access-control Requirements" and Section 34.20, "Enclave Page Cache Map (EPCM)" in Chapter 34, "Enclave Access Control and Data Structures." Such an exception is called an **SGX-induced page fault**. The processor uses the error code to distinguish SGX-induced page faults from ordinary page faults.

Figure 4-12 illustrates the error code that the processor provides on delivery of a page-fault exception. The following items explain how the bits in the error code describe the nature of the page-fault exception:



P
0 The fault was caused by a non-present page.
1 The fault was caused by a page-level protection violation.

W/R
0 The access causing the fault was a read.
1 The access causing the fault was a write.

U/S
0 A supervisor-mode access caused the fault.
1 A user-mode access caused the fault.

RSVD
0 The fault was not caused by reserved bit violation.
1 The fault was caused by a reserved bit set to 1 in some paging-structure entry.

I/D
0 The fault was not caused by an instruction fetch.
1 The fault was caused by an instruction fetch.

PK
0 The fault was not caused by protection keys.
1 There was a protection-key violation.

SS
0 The fault was not caused by a shadow-stack access.
1 The fault was caused by a shadow-stack access.

HLAT
0 The fault occurred during ordinary paging or due to access rights.
1 The fault occurred during HLAT paging.

SGX
0 The fault is not related to SGX.
1 The fault resulted from violation of SGX-specific access-control requirements.

Figure 4-12. Page-Fault Error Code

- 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.

1. If HLAT paging encounters a paging-structure entry that sets a reserved bit, there is no translation even if the bit 11 of the entry indicates a restart. In this case, there is a page fault and the translation is not restarted.

- 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. User-mode and supervisor-mode accesses are defined in Section 4.6.
- 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. (Because reserved bits are not checked in a paging-structure entry whose P flag is 0, bit 3 of the error code can be set only if bit 0 is also set.[1])

  Bits reserved in the paging-structure entries are reserved for future functionality. Software developers should be aware that such bits may be used in the future and that a paging-structure entry that causes a page-fault exception on one processor might not do so in the future.
- I/D flag (bit 4).
  This flag is 1 if (1) the access causing the page-fault exception was an instruction fetch; and (2) either (a) CR4.SMEP = 1; or (b) both (i) CR4.PAE = 1 (either PAE paging, 4-level paging, or 5-level paging is in use); and (ii) IA32_EFER.NXE = 1. Otherwise, the flag is 0. 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 only for data accesses and only with 4-level paging and 5-level paging. In these cases, the setting depends on the mode of the address being accessed:
  - For accesses to supervisor-mode addresses, the flag is set if (1) CR4.PKS = 1; (2) the linear address has protection key *i*; and (3) the IA32_PKRS MSR (see Section 4.6.2) is such that either (a) AD*i* = 1; or (b) the following all hold: (i) WD*i* = 1; (ii) the access is a write access; and (iii) either CR0.WP = 1 or the access causing the page-fault exception was a user-mode access. (Note that this flag may be set on page faults due to user-mode accesses to supervisor-mode addresses.)
  - For accesses to user-mode addresses, the flag is set if (1) CR4.PKE = 1; (2) the linear address has protection key *i*; and (3) the PKRU register (see Section 4.6.2) is such that either (a) AD*i* = 1; or (b) the following all hold: (i) WD*i* = 1; (ii) the access is a write access; and (iii) either CR0.WP = 1 or the access causing the page-fault exception was a user-mode 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.

Page-fault exceptions occur only due to an attempt to use a linear address. Failures to load the PDPTE registers with PAE paging (see Section 4.4.1) cause general-protection exceptions (#GP(0)) and not page-fault exceptions.

1. Some past processors had errata for some page faults that occur when 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. Due to these errata, some such page faults produced error codes that cleared bit 0 (P flag) and set bit 3 (RSVD flag).

4.8 ACCESSED AND DIRTY FLAGS

For any paging-structure entry that is used during linear-address translation, bit 5 is the **accessed** flag.[1] For paging-structure entries that map a page (as opposed to referencing another paging structure), bit 6 is the **dirty** flag. These flags are provided for use by memory-management software to manage the transfer of pages and paging structures into and out of physical memory.

Whenever the processor uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set).

Whenever there is a write to a linear address, the processor sets the dirty flag (if it is not already set) in the paging-structure entry that identifies the final physical address for the linear address (either a PTE or a paging-structure entry in which the PS flag is 1).

The previous two paragraphs apply also to HLAT paging. If HLAT paging encounters a paging-structure entry that sets bit 11, indicating a restart, the processor will set the accessed flag in that entry; it will not set the dirty flag because, if an entry indicates a restart, it does identify the final physical address for the linear address being translated.

NOTE

If software on one logical processor writes to a page while software on another logical processor concurrently clears the R/W flag in the paging-structure entry that maps the page, execution on some processors may result in the entry's dirty flag being set (due to the write on the first logical processor) and the entry's R/W flag being clear (due to the update to the entry on the second logical processor). This will never occur on a processor that supports control-flow enforcement technology (CET). Specifically, a processor that supports CET will never set the dirty flag in a paging-structure entry in which the R/W flag is clear.

Memory-management software may clear these flags when a page or a paging structure is initially loaded into physical memory. These flags are "sticky," meaning that, once set, the processor does not clear them; only software can clear them.

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). This fact implies that, if software changes an accessed flag or a dirty flag from 1 to 0, the processor might not set the corresponding bit in memory on a subsequent access using an affected linear address (see Section 4.10.4.3). See Section 4.10.4.2 for how software can ensure that these bits are updated as desired.

NOTE

The accesses used by the processor to set these flags may or may not be exposed to the processor's self-modifying code detection logic. If the processor is executing code from the same memory area that is being used for the paging structures, the setting of these flags may or may not result in an immediate change to the executing code stream.

4.9 PAGING AND MEMORY TYPING

The **memory type** of a memory access refers to the type of caching used for that access. Chapter 11, "Memory Cache Control" provides many details regarding memory typing in the Intel-64 and IA-32 architectures. This section describes how paging contributes to the determination of memory typing.

The way in which paging contributes to memory typing depends on whether the processor supports the **Page Attribute Table** (**PAT**; see Section 11.12).[2] Section 4.9.1 and Section 4.9.2 explain how paging contributes to memory typing depending on whether the PAT is supported.

1. With PAE paging, the PDPTEs are not used during linear-address translation but only to load the PDPTE registers for some executions of the MOV CR instruction (see Section 4.4.1). For this reason, the PDPTEs do not contain accessed flags with PAE paging.
2. The PAT is supported on Pentium III and more recent processor families. See Section 4.1.4 for how to determine whether the PAT is supported.

4.9.1 Paging and Memory Typing When the PAT is Not Supported (Pentium Pro and Pentium II Processors)

NOTE

The PAT is supported on all processors that support 4-level paging or 5-level paging. Thus, this section applies only to 32-bit paging and PAE paging.

If the PAT is not supported, paging contributes to memory typing in conjunction with the memory-type range registers (MTRRs) as specified in Table 11-6 in Section 11.5.2.1.

For any access to a physical address, the table combines the memory type specified for that physical address by the MTRRs with a PCD value and a PWT value. The latter two values are determined as follows:

- For an access to a PDE with 32-bit paging, the PCD and PWT values come from CR3.
- For an access to a PDE with PAE paging, the PCD and PWT values come from the relevant PDPTE register.
- For an access to a PTE, the PCD and PWT values come from the relevant PDE.
- For an access to the physical address that is the translation of a linear address, the PCD and PWT values come from the relevant PTE (if the translation uses a 4-KByte page) or the relevant PDE (otherwise).
- With PAE paging, the UC memory type is used when loading the PDPTEs (see Section 4.4.1).

4.9.2 Paging and Memory Typing When the PAT is Supported (Pentium III and More Recent Processor Families)

If the PAT is supported, paging contributes to memory typing in conjunction with the PAT and the memory-type range registers (MTRRs) as specified in Table 11-7 in Section 11.5.2.2.

The PAT is a 64-bit MSR (IA32_PAT; MSR index 277H) comprising eight (8) 8-bit entries (entry *i* comprises bits 8*i*+7:8*i* of the MSR).

For any access to a physical address, the table combines the memory type specified for that physical address by the MTRRs with a memory type selected from the PAT. Table 11-11 in Section 11.12.3 specifies how a memory type is selected from the PAT. Specifically, it comes from entry *i* of the PAT, where *i* is defined as follows:

- For an access to an entry in a paging structure whose address is in CR3 (e.g., the PML4 table with 4-level paging):
  - For 4-level paging or 5-level paging with CR4.PCIDE = 1, *i* = 0.
  - Otherwise, *i* = 2*PCD+PWT, where the PCD and PWT values come from CR3.
- For an access to a PDE with PAE paging, *i* = 2*PCD+PWT, where the PCD and PWT values come from the relevant PDPTE register.
- For an access to a paging-structure entry X whose address is in another paging-structure entry Y, *i* = 2*PCD+PWT, where the PCD and PWT values come from Y.
- For an access to the physical address that is the translation of a linear address, *i* = 4*PAT+2*PCD+PWT, where the PAT, PCD, and PWT values come from the relevant PTE (if the translation uses a 4-KByte page), the relevant PDE (if the translation uses a 2-MByte page or a 4-MByte page), or the relevant PDPTE (if the translation uses a 1-GByte page).
- With PAE paging, the WB memory type is used when loading the PDPTEs (see Section 4.4.1).[1]

1. Some older IA-32 processors used the UC memory type when loading the PDPTEs. Some processors may use the UC memory type if CR0.CD = 1 or if the MTRRs are disabled. These behaviors are model-specific and not architectural.

4.9.3 Caching Paging-Related Information about Memory Typing

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). These structures may include information about memory typing. The processor may use memory-typing information from the TLBs and paging-structure caches instead of from the paging structures in memory.

This fact implies that, if software modifies a paging-structure entry to change the memory-typing bits, the processor might not use that change for a subsequent translation using that entry or for access to an affected linear address. See Section 4.10.4.2 for how software can ensure that the processor uses the modified memory typing.

4.10 CACHING TRANSLATION INFORMATION

The Intel-64 and IA-32 architectures may accelerate the address-translation process by caching data from the paging structures on the processor. Because the processor does not ensure that the data that it caches are always consistent with the structures in memory, it is important for software developers to understand how and when the processor may cache such data. They should also understand what actions software can take to remove cached data that may be inconsistent and when it should do so. This section provides software developers information about the relevant processor operation.

Section 4.10.1 introduces process-context identifiers (PCIDs), which a logical processor may use to distinguish information cached for different linear-address spaces. Section 4.10.2 and Section 4.10.3 describe how the processor may cache information in translation lookaside buffers (TLBs) and paging-structure caches, respectively. Section 4.10.4 explains how software can remove inconsistent cached information by invalidating portions of the TLBs and paging-structure caches. Section 4.10.5 describes special considerations for multiprocessor systems.

4.10.1 Process-Context Identifiers (PCIDs)

Process-context identifiers (**PCIDs**) are a facility by which a logical processor may cache information for multiple linear-address spaces. The processor may retain cached information when software switches to a different linear-address space with a different PCID (e.g., by loading CR3; see Section 4.10.4.1 for details).

A PCID is a 12-bit identifier. Non-zero PCIDs are enabled by setting the PCIDE flag (bit 17) of CR4. If CR4.PCIDE = 0, the current PCID is always 000H; otherwise, the current PCID is the value of bits 11:0 of CR3.[1] Not all processors allow CR4.PCIDE to be set to 1; see Section 4.1.4 for how to determine whether this is allowed.

The processor ensures that CR4.PCIDE can be 1 only in IA-32e mode (thus, 32-bit paging and PAE paging use only PCID 000H). In addition, software can change CR4.PCIDE from 0 to 1 only if CR3[11:0] = 000H. These requirements are enforced by the following limitations on the MOV CR instruction:

- MOV to CR4 causes a general-protection exception (#GP) if it would change CR4.PCIDE from 0 to 1 and either IA32_EFER.LMA = 0 or CR3[11:0] ≠ 000H.
- MOV to CR0 causes a general-protection exception if it would clear CR0.PG to 0 while CR4.PCIDE = 1.

When a logical processor creates entries in the TLBs (Section 4.10.2) and paging-structure caches (Section 4.10.3), it associates those entries with the current PCID. When using entries in the TLBs and paging-structure caches to translate a linear address, a logical processor uses only those entries associated with the current PCID (see Section 4.10.2.4 for an exception).

If CR4.PCIDE = 0, a logical processor does not cache information for any PCID other than 000H. This is because (1) if CR4.PCIDE = 0, the logical processor will associate any newly cached information with the current PCID, 000H; and (2) if MOV to CR4 clears CR4.PCIDE, all cached information is invalidated (see Section 4.10.4.1).

1. Note that, while HLAT paging (Section 4.5.3) does not use CR3 to locate the first paging structure, it does use the PCID in CR3[11:0] when CR4.PCIDE = 1.

NOTE

In revisions of this manual that were produced when no processors allowed CR4.PCIDE to be set to 1, the Section "Caching Translation Information" discussed the caching of translation information without any reference to PCIDs. While the section now refers to PCIDs in its specification of this caching, this documentation change is not intended to imply any change to the behavior of processors that do not allow CR4.PCIDE to be set to 1.

4.10.2 Translation Lookaside Buffers (TLBs)

A processor may cache information about the translation of linear addresses in translation lookaside buffers (TLBs). In general, TLBs contain entries that map page numbers to page frames; these terms are defined in Section 4.10.2.1. Section 4.10.2.2 describes how information may be cached in TLBs, and Section 4.10.2.3 gives details of TLB usage. Section 4.10.2.4 explains the global-page feature, which allows software to indicate that certain translations should receive special treatment when cached in the TLBs.

4.10.2.1 Page Numbers, Page Frames, and Page Offsets

Section 4.3, Section 4.4.2, and Section 4.5 give details of how the different paging modes translate linear addresses to physical addresses. Specifically, the upper bits of a linear address (called the **page number**) determine the upper bits of the physical address (called the **page frame**); the lower bits of the linear address (called the **page offset**) determine the lower bits of the physical address. The boundary between the page number and the page offset is determined by the **page size**. Specifically:

- 32-bit paging:
  - If the translation does not use a PTE (because CR4.PSE = 1 and the PS flag is 1 in the PDE used), the page size is 4 MBytes and the page number comprises bits 31:22 of the linear address.
  - If the translation does use a PTE, the page size is 4 KBytes and the page number comprises bits 31:12 of the linear address.
- PAE paging:
  - If the translation does not use a PTE (because the PS flag is 1 in the PDE used), the page size is 2 MBytes and the page number comprises bits 31:21 of the linear address.
  - If the translation does use a PTE, the page size is 4 KBytes and the page number comprises bits 31:12 of the linear address.
- 4-level paging and 5-level paging:
  - If the translation does not use a PDE (because the PS flag is 1 in the PDPTE used), the page size is 1 GByte and the page number comprises bits 47:30 of the linear address.
  - If the translation does use a PDE but does not uses a PTE (because the PS flag is 1 in the PDE used), the page size is 2 MBytes and the page number comprises bits 47:21 of the linear address.
  - If the translation does use a PTE, the page size is 4 KBytes and the page number comprises bits 47:12 of the linear address.
  - The page size identified by the preceding items may be reduced if there has been a restart of HLAT paging (see Section 4.5.5). Restart of HLAT paging always specifies a maximum page size; this page size is determined by the level of the paging-structure entry that caused the restart. The page size used by the translation is the minimum of the maximum page size specified by the restart and the page size determined by the restarted translation (as specified by the previous items).

    For example, suppose that HLAT paging encounters a PDE that sets bit 11, indicating a restart. As a result, the restart uses a maximum page size of 2 MBytes. Suppose that the restarted translation encounters a PDPTE that sets bit 7, indicating a 1-GByte page. In this case, the translation produced will have a page size of 2 MBytes (the smaller of the two sizes).

4.10.2.2 Caching Translations in TLBs

The processor may accelerate the paging process by caching individual translations in **translation lookaside buffers** (**TLBs**). Each entry in a TLB is an individual translation. Each translation is referenced by a page number. It contains the following information from the paging-structure entries used to translate linear addresses with the page number:

- The physical address corresponding to the page number (the page frame).
- The access rights from the paging-structure entries used to translate linear addresses with the page number (see Section 4.6):
  - The logical-AND of the R/W flags.
  - The logical-AND of the U/S flags.
  - The logical-OR of the XD flags (necessary only if IA32_EFER.NXE = 1).
  - The protection key (only with 4-level paging and 5-level paging).
- Attributes from a paging-structure entry that identifies the final page frame for the page number (either a PTE or a paging-structure entry in which the PS flag is 1):
  - The dirty flag (see Section 4.8).
  - The memory type (see Section 4.9).

(TLB entries may contain other information as well. A processor may implement multiple TLBs, and some of these may be for special purposes, e.g., only for instruction fetches. Such special-purpose TLBs may not contain some of this information if it is not necessary. For example, a TLB used only for instruction fetches need not contain information about the R/W and dirty flags.)

As noted in Section 4.10.1, any TLB entries created by a logical processor are associated with the current PCID.

Processors need not implement any TLBs. Processors that do implement TLBs may invalidate any TLB entry at any time. Software should not rely on the existence of TLBs or on the retention of TLB entries.

4.10.2.3 Details of TLB Use

Because the TLBs cache entries only for linear addresses with translations, there can be a TLB entry for a page number only if the P flag is 1 and the reserved bits are 0 in each of the paging-structure entries used to translate that page number. In addition, the processor does not cache a translation for a page number unless the accessed flag is 1 in each of the paging-structure entries used during translation; before caching a translation, the processor sets any of these accessed flags that is not already 1.

Subject to the limitations given in the previous paragraph, the processor may cache a translation for any linear address, even if that address is not used to access memory. For example, the processor may cache translations required for prefetches and for accesses that result from speculative execution that would never actually occur in the executed code path.

If the page number of a linear address corresponds to a TLB entry associated with the current PCID, the processor may use that TLB entry to determine the page frame, access rights, and other attributes for accesses to that linear address. In this case, the processor may not actually consult the paging structures in memory. The processor may retain a TLB entry unmodified even if software subsequently modifies the relevant paging-structure entries in memory. See Section 4.10.4.2 for how software can ensure that the processor uses the modified paging-structure entries.

If the paging structures specify a translation using a page larger than 4 KBytes, some processors may cache multiple smaller-page TLB entries for that translation. Each such TLB entry would be associated with a page number corresponding to the smaller page size (e.g., bits 47:12 of a linear address with 4-level paging), even though part of that page number (e.g., bits 20:12) is part of the offset with respect to the page specified by the paging structures. The upper bits of the physical address in such a TLB entry are derived from the physical address in the PDE used to create the translation, while the lower bits come from the linear address of the access for which the translation is created. There is no way for software to be aware that multiple translations for smaller pages have been used for a large page. For example, an execution of INVLPG for a linear address on such a page invalidates any and all smaller-page TLB entries for the translation of any linear address on that page.

If software modifies the paging structures so that the page size used for a 4-KByte range of linear addresses changes, the TLBs may subsequently contain multiple translations for the address range (one for each page size). A reference to a linear address in the address range may use any of these translations. Which translation is used may vary from one execution to another, and the choice may be implementation-specific.

4.10.2.4 Global Pages

The Intel-64 and IA-32 architectures also allow for **global pages** when the PGE flag (bit 7) is 1 in CR4. If the G flag (bit 8) is 1 in a paging-structure entry that maps a page (either a PTE or a paging-structure entry in which the PS flag is 1), any TLB entry cached for a linear address using that paging-structure entry is considered to be **global**. Because the G flag is used only in paging-structure entries that map a page, and because information from such entries is not cached in the paging-structure caches, the global-page feature does not affect the behavior of the paging-structure caches.

A logical processor may use a global TLB entry to translate a linear address, even if the TLB entry is associated with a PCID different from the current PCID.

4.10.3 Paging-Structure Caches

In addition to the TLBs, a processor may cache other information about the paging structures in memory.

4.10.3.1 Caches for Paging Structures

A processor may support any or all of the following paging-structure caches:

- **PML5E cache** (5-level paging only). Each PML5E-cache entry is referenced by a 9-bit value and is used for linear addresses for which bits 56:40 have that value. The entry contains information from the PML5E used to translate such linear addresses:
  - The physical address from the PML5E (the address of the PML4 table).
  - The value of the R/W flag of the PML5E.
  - The value of the U/S flag of the PML5E.
  - The value of the XD flag of the PML5E.
  - The values of the PCD and PWT flags of the PML5E.

  The following items detail how a processor may use the PML5E cache:
  - If the processor has a PML5E-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E in memory).
  - The processor does not create a PML5E-cache entry unless the P flag is 1 and all reserved bits are 0 in the PML5E in memory.
  - The processor does not create a PML5E-cache entry unless the accessed flag is 1 in the PML5E in memory; before caching a translation, the processor sets the accessed flag if it is not already 1.
  - The processor may create a PML5E-cache entry even if there are no translations for any linear address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced PML4 table).
  - If the processor creates a PML5E-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML5E in memory.
- **PML4E cache** (4-level paging and 5-level paging only). The use of the PML4E cache depends on the paging mode:
  - For 4-level paging, each PML4E-cache entry is referenced by a 9-bit value and is used for linear addresses for which bits 47:39 have that value.
  - For 5-level paging, each PML4E-cache entry is referenced by an 18-bit value and is used for linear addresses for which bits 56:39 have that value.

  A PML4E-cache entry contains information from the PML5E and PML4E used to translate the relevant linear addresses (for 4-level paging, the PML5E does not apply):

— The physical address from the PML4E (the address of the page-directory-pointer table).

— The logical-AND of the R/W flags in the PML5E and the PML4E.

— The logical-AND of the U/S flags in the PML5E and the PML4E.

— The logical-OR of the XD flags in the PML5E and the PML4E.

— The values of the PCD and PWT flags of the PML4E.

The following items detail how a processor may use the PML4E cache:

— If the processor has a PML4E-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E and PML4E in memory).

— The processor does not create a PML4E-cache entry unless the P flags are 1 and all reserved bits are 0 in the PML5E and the PML4E in memory.

— The processor does not create a PML4E-cache entry unless the accessed flags are 1 in the PML5E and the PML4E in memory; before caching a translation, the processor sets any accessed flags that are not already 1.

— The processor may create a PML4E-cache entry even if there are no translations for any linear address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced page-directory-pointer table).

— If the processor creates a PML4E-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML4E in memory.

- **PDPTE cache** (4-level paging and 5-level paging only).[1] The use of the PML4E cache depends on the paging mode:

  — For 4-level paging, each PDPTE-cache entry is referenced by an 18-bit value and is used for linear addresses for which bits 47:30 have that value.

  — For 5-level paging, each PDPTE-cache entry is referenced by a 27-bit value and is used for linear addresses for which bits 56:30 have that value.

  A PDPTE-cache entry contains information from the PML5E, PML4E, PDPTE used to translate the relevant linear addresses (for 4-level paging, the PML5E does not apply):

  — The physical address from the PDPTE (the address of the page directory). (No PDPTE-cache entry is created for a PDPTE that maps a 1-GByte page.)

  — The logical-AND of the R/W flags in the PML5E, PML4E, and PDPTE.

  — The logical-AND of the U/S flags in the PML5E, PML4E, and PDPTE.

  — The logical-OR of the XD flags in the PML5E, PML4E, and PDPTE.

  — The values of the PCD and PWT flags of the PDPTE.

  The following items detail how a processor may use the PDPTE cache:

  — If the processor has a PDPTE-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E, PML4E, and PDPTE in memory).

  — The processor does not create a PDPTE-cache entry unless the P flags are 1, the PS flags are 0, and the reserved bits are 0 in the PML5E, PML4E, and PDPTE in memory.

  — The processor does not create a PDPTE-cache entry unless the accessed flags are 1 in the PML5E, PML4E and PDPTE in memory; before caching a translation, the processor sets any accessed flags that are not already 1.

  — The processor may create a PDPTE-cache entry even if there are no translations for any linear address that might use that entry.

  — If the processor creates a PDPTE-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML5E, PML4E, or PDPTE in memory.

1. With PAE paging, the PDPTEs are stored in internal, non-architectural registers. The operation of these registers is described in Section 4.4.1 and differs from that described here.

- **PDE cache**. The use of the PDE cache depends on the paging mode:
  - For 32-bit paging, each PDE-cache entry is referenced by a 10-bit value and is used for linear addresses for which bits 31:22 have that value.
  - For PAE paging, each PDE-cache entry is referenced by an 11-bit value and is used for linear addresses for which bits 31:21 have that value.
  - For 4-level paging, each PDE-cache entry is referenced by a 27-bit value and is used for linear addresses for which bits 47:21 have that value.
  - For 5-level paging, each PDE-cache entry is referenced by a 36-bit value and is used for linear addresses for which bits 56:21 have that value.

  A PDE-cache entry contains information from the PML5E, PML4E, PDPTE, and PDE used to translate the relevant linear addresses (for 32-bit paging and PAE paging, only the PDE applies; for 4-level paging, the PML5E does not apply):
  - The physical address from the PDE (the address of the page table). (No PDE-cache entry is created for a PDE that maps a page.)
  - The logical-AND of the R/W flags in the PML5E, PML4E, PDPTE, and PDE.
  - The logical-AND of the U/S flags in the PML5E, PML4E, PDPTE, and PDE.
  - The logical-OR of the XD flags in the PML5E, PML4E, PDPTE, and PDE.
  - The values of the PCD and PWT flags of the PDE.

  The following items detail how a processor may use the PDE cache (references below to PML5Es, PML4Es, and PDPTEs apply only to 4-level paging and to 5-level paging, as appropriate):
  - If the processor has a PDE-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E, PML4E, PDPTE, and PDE in memory).
  - The processor does not create a PDE-cache entry unless the P flags are 1, the PS flags are 0, and the reserved bits are 0 in the PML5E, PML4E, PDPTE, and PDE in memory.
  - The processor does not create a PDE-cache entry unless the accessed flag is 1 in the PML5E, PML4E, PDPTE, and PDE in memory; before caching a translation, the processor sets any accessed flags that are not already 1.
  - The processor may create a PDE-cache entry even if there are no translations for any linear address that might use that entry.
  - If the processor creates a PDE-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML5E, PML4E, PDPTE, or PDE in memory.

Information from a paging-structure entry can be included in entries in the paging-structure caches for other paging-structure entries referenced by the original entry. For example, if the R/W flag is 0 in a PML4E, then the R/W flag will be 0 in any PDPTE-cache entry for a PDPTE from the page-directory-pointer table referenced by that PML4E. This is because the R/W flag of each such PDPTE-cache entry is the logical-AND of the R/W flags in the appropriate PML4E and PDPTE.

On processors that support HLAT paging (see Section 4.5.1), each entry in a paging-structure cache indicates whether the entry was cached during ordinary paging or HLAT paging. When the processor commences linear-address translation using ordinary paging (respectively, HLAT paging), it will use only entries that indicate that they were cached during ordinary paging (respectively, HLAT paging).

Entries that were cached during HLAT paging also include the restart flag (bit 11) of the original paging-structure entry. When the processor commences HLAT paging using such an entry, it immediately restarts (using ordinary paging) if this cached restart flag is 1.

The paging-structure caches contain information only from paging-structure entries that reference other paging structures (and not those that map pages). Because the G flag is not used in such paging-structure entries, the global-page feature does not affect the behavior of the paging-structure caches.

The processor may create entries in paging-structure caches for translations required for prefetches and for accesses that are a result of speculative execution that would never actually occur in the executed code path.

As noted in Section 4.10.1, any entries created in paging-structure caches by a logical processor are associated with the current PCID.

A processor may or may not implement any of the paging-structure caches. Software should rely on neither their presence nor their absence. The processor may invalidate entries in these caches at any time. Because the processor may create the cache entries at the time of translation and not update them following subsequent modifications to the paging structures in memory, software should take care to invalidate the cache entries appropriately when causing such modifications. The invalidation of TLBs and the paging-structure caches is described in Section 4.10.4.

4.10.3.2 Using the Paging-Structure Caches to Translate Linear Addresses

When a linear address is accessed, the processor uses a procedure such as the following to determine the physical address to which it translates and whether the access should be allowed:

- If the processor finds a TLB entry that is for the page number of the linear address and that is associated with the current PCID (or which is global), it may use the physical address, access rights, and other attributes from that entry.
- If the processor does not find a relevant TLB entry, it may use the upper bits of the linear address to select an entry from the PDE cache that is associated with the current PCID (Section 4.10.3.1 indicates which bits are used in each paging mode). It can then use that entry to complete the translation process (locating a PTE, etc.) as if it had traversed the PDE (and, for 4-level paging and 5-level paging, the PDPTE, PML4E, and PML5E, as appropriate) corresponding to the PDE-cache entry.
- The following items apply when 4-level paging or 5-level paging is used:
  - If the processor does not find a relevant TLB entry or PDE-cache entry, it may use the upper bits of the linear address (for 4-level paging, bits 47:30; for 5-level paging, bits 56:30) to select an entry from the PDPTE cache that is associated with the current PCID. It can then use that entry to complete the translation process (locating a PDE, etc.) as if it had traversed the PDPTE, the PML4E, and (for 5-level paging) the PML5E corresponding to the PDPTE-cache entry.
  - If the processor does not find a relevant TLB entry, PDE-cache entry, or PDPTE-cache entry, it may use the upper bits of the linear address (for 4-level paging, bits 47:39; for 5-level paging, bits 56:39) to select an entry from the PML4E cache that is associated with the current PCID. It can then use that entry to complete the translation process (locating a PDPTE, etc.) as if it had traversed the corresponding PML4E.
  - With 5-level paging, if the processor does not find a relevant TLB entry, PDE-cache entry, PDPTE-cache entry, or PML4E-cache entry, it may use bits 56:48 of the linear address to select an entry from the PML5E cache that is associated with the current PCID. It can then use that entry to complete the translation process (locating a PML4E, etc.) as if it had traversed the corresponding PML5E.

(Any of the above steps would be skipped if the processor does not support the cache in question.)

If the processor does not find a TLB or paging-structure-cache entry for the linear address, it uses the linear address to traverse the entire paging-structure hierarchy, as described in Section 4.3, Section 4.4.2, and Section 4.5.

4.10.3.3 Multiple Cached Entries for a Single Paging-Structure Entry

The paging-structure caches and TLBs may contain multiple entries associated with a single PCID and with information derived from a single paging-structure entry. The following items give some examples for 4-level paging:

- Suppose that two PML4Es contain the same physical address and thus reference the same page-directory-pointer table. Any PDPTE in that table may result in two PDPTE-cache entries, each associated with a different set of linear addresses. Specifically, suppose that the n_1^{th} and n_2^{th} entries in the PML4 table contain the same physical address. This implies that the physical address in the m^{th} PDPTE in the page-directory-pointer table would appear in the PDPTE-cache entries associated with both p_1 and p_2, where $(p_1 \gg 9) = n_1$, $(p_2 \gg 9) = n_2$, and $(p_1 \& 1FFH) = (p_2 \& 1FFH) = m$. This is because both PDPTE-cache entries use the same PDPTE, one resulting from a reference from the n_1^{th} PML4E and one from the n_2^{th} PML4E.
- Suppose that the first PML4E (i.e., the one in position 0) contains the physical address X in CR3 (the physical address of the PML4 table). This implies the following:

— Any PML4-cache entry associated with linear addresses with 0 in bits 47:39 contains address X.

— Any PDPTE-cache entry associated with linear addresses with 0 in bits 47:30 contains address X. This is because the translation for a linear address for which the value of bits 47:30 is 0 uses the value of bits 47:39 (0) to locate a page-directory-pointer table at address X (the address of the PML4 table). It then uses the value of bits 38:30 (also 0) to find address X again and to store that address in the PDPTE-cache entry.

— Any PDE-cache entry associated with linear addresses with 0 in bits 47:21 contains address X for similar reasons.

— Any TLB entry for page number 0 (associated with linear addresses with 0 in bits 47:12) translates to page frame X » 12 for similar reasons.

The same PML4E contributes its address X to all these cache entries because the self-referencing nature of the entry causes it to be used as a PML4E, a PDPTE, a PDE, and a PTE.

4.10.4 Invalidation of TLBs and Paging-Structure Caches

As noted in Section 4.10.2 and Section 4.10.3, the processor may create entries in the TLBs and the paging-structure caches when linear addresses are translated, and it may retain these entries even after the paging structures used to create them have been modified. To ensure that linear-address translation uses the modified paging structures, software should take action to invalidate any cached entries that may contain information that has since been modified.

4.10.4.1 Operations that Invalidate TLBs and Paging-Structure Caches

The following instructions invalidate entries in the TLBs and the paging-structure caches:

- INVLPG. This instruction takes a single operand, which is a linear address. The instruction invalidates any TLB entries that are for a page number corresponding to the linear address and that are associated with the current PCID. It also invalidates any global TLB entries with that page number, regardless of PCID (see Section 4.10.2.4).[1] INVLPG also invalidates all entries in all paging-structure caches associated with the current PCID, regardless of the linear addresses to which they correspond.
- INVPCID. The operation of this instruction is based on instruction operands, called the INVPCID type and the INVPCID descriptor. Four INVPCID types are currently defined:
  — Individual-address. If the INVPCID type is 0, the logical processor invalidates mappings—except global translations—associated with the PCID specified in the INVPCID descriptor and that would be used to translate the linear address specified in the INVPCID descriptor.[2] (The instruction may also invalidate global translations, as well as mappings associated with other PCIDs and for other linear addresses.)
  — Single-context. If the INVPCID type is 1, the logical processor invalidates all mappings—except global translations—associated with the PCID specified in the INVPCID descriptor. (The instruction may also invalidate global translations, as well as mappings associated with other PCIDs.)
  — All-context, including globals. If the INVPCID type is 2, the logical processor invalidates mappings—including global translations—associated with all PCIDs.
  — All-context. If the INVPCID type is 3, the logical processor invalidates mappings—except global translations—associated with all PCIDs. (The instruction may also invalidate global translations.)

  See Chapter 3 of the *Intel 64 and IA-32 Architecture Software Developer's Manual, Volume 2A* for details of the INVPCID instruction.
- MOV to CR0. The instruction invalidates all TLB entries (including global entries) and all entries in all paging-structure caches (for all PCIDs) if it changes the value of CR0.PG from 1 to 0.
- MOV to CR3. The behavior of the instruction depends on the value of CR4.PCIDE:

1. If the paging structures map the linear address using a page larger than 4 KBytes and there are multiple TLB entries for that page (see Section 4.10.2.3), the instruction invalidates all of them.
2. If the paging structures map the linear address using a page larger than 4 KBytes and there are multiple TLB entries for that page (see Section 4.10.2.3), the instruction invalidates all of them.

  - If CR4.PCIDE = 0, the instruction invalidates all TLB entries associated with PCID 000H except those for global pages. It also invalidates all entries in all paging-structure caches associated with PCID 000H.
  - If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 0, the instruction invalidates all TLB entries associated with the PCID specified in bits 11:0 of the instruction's source operand except those for global pages. It also invalidates all entries in all paging-structure caches associated with that PCID. It is not required to invalidate entries in the TLBs and paging-structure caches that are associated with other PCIDs.
  - If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 1, the instruction is not required to invalidate any TLB entries or entries in paging-structure caches.
- MOV to CR4. The behavior of the instruction depends on the bits being modified:
  - The instruction invalidates all TLB entries (including global entries) and all entries in all paging-structure caches (for all PCIDs) if (1) it changes the value of CR4.PGE;[1] or (2) it changes the value of the CR4.PCIDE from 1 to 0.
  - The instruction invalidates all TLB entries and all entries in all paging-structure caches for the current PCID if (1) it changes the value of CR4.PAE; or (2) it changes the value of CR4.SMEP from 0 to 1.
- Task switch. If a task switch changes the value of CR3, it invalidates all TLB entries associated with PCID 000H except those for global pages. It also invalidates all entries in all paging-structure caches associated with PCID 000H.[2]
- VMX transitions. See Section 4.11.1.

The processor is always free to invalidate additional entries in the TLBs and paging-structure caches. The following are some examples:

- INVLPG may invalidate TLB entries for pages other than the one corresponding to its linear-address operand. It may invalidate TLB entries and paging-structure-cache entries associated with PCIDs other than the current PCID.
- INVPCID may invalidate TLB entries for pages other than the one corresponding to the specified linear address. It may invalidate TLB entries and paging-structure-cache entries associated with PCIDs other than the specified PCID.
- MOV to CR0 may invalidate TLB entries even if CR0.PG is not changing. For example, this may occur if either CR0.CD or CR0.NW is modified.
- MOV to CR3 may invalidate TLB entries for global pages. If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 0, it may invalidate TLB entries and entries in the paging-structure caches associated with PCIDs other than the PCID it is establishing. It may invalidate entries if CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 1.
- MOV to CR4 may invalidate TLB entries when changing CR4.PSE or when changing CR4.SMEP from 1 to 0.
- On a processor supporting Hyper-Threading Technology, invalidations performed on one logical processor may invalidate entries in the TLBs and paging-structure caches used by other logical processors.

(Other instructions and operations may invalidate entries in the TLBs and the paging-structure caches, but the instructions identified above are recommended.)

In addition to the instructions identified above, page faults invalidate entries in the TLBs and paging-structure caches. In particular, a page-fault exception resulting from an attempt to use a linear address will invalidate any TLB entries that are for a page number corresponding to that linear address and that are associated with the current PCID. It also invalidates all entries in the paging-structure caches that would be used for that linear address and that are associated with the current PCID.[3] These invalidations ensure that the page-fault exception will not recur (if the faulting instruction is re-executed) if it would not be caused by the contents of the paging structures

1. If CR4.PGE is changing from 0 to 1, there were no global TLB entries before the execution; if CR4.PGE is changing from 1 to 0, there will be no global TLB entries after the execution.
2. Task switches do not occur in IA-32e mode and thus cannot occur with 4-level paging. Since CR4.PCIDE can be set only with 4-level paging, task switches occur only with CR4.PCIDE = 0.
3. Unlike INVLPG, page faults need not invalidate **all** entries in the paging-structure caches, only those that would be used to translate the faulting linear address.

in memory (and if, therefore, it resulted from cached entries that were not invalidated after the paging structures were modified in memory).

As noted in Section 4.10.2, some processors may choose to cache multiple smaller-page TLB entries for a translation specified by the paging structures to use a page larger than 4 KBytes. There is no way for software to be aware that multiple translations for smaller pages have been used for a large page. The INVLPG instruction and page faults provide the same assurances that they provide when a single TLB entry is used: they invalidate all TLB entries corresponding to the translation specified by the paging structures.

4.10.4.2 Recommended Invalidation

The following items provide some recommendations regarding when software should perform invalidations:

- If software modifies a paging-structure entry that maps a page (rather than referencing another paging structure), it should execute INVLPG for any linear address with a page number whose translation uses that paging-structure entry.[1]

  (If the paging-structure entry may be used in the translation of different page numbers — see Section 4.10.3.3 — software should execute INVLPG for linear addresses with each of those page numbers; alternatively, it could use MOV to CR3 or MOV to CR4.)
- If software modifies a paging-structure entry that references another paging structure, it may use one of the following approaches depending upon the types and number of translations controlled by the modified entry:
  - — Execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry. However, if no page numbers that would use the entry have translations (e.g., because the P flags are 0 in all entries in the paging structure referenced by the modified entry), it remains necessary to execute INVLPG at least once.
  - — Execute MOV to CR3 if the modified entry controls no global pages.
  - — Execute MOV to CR4 to modify CR4.PGE.
- If CR4.PCIDE = 1 and software modifies a paging-structure entry that does not map a page or in which the G flag (bit 8) is 0, additional steps are required if the entry may be used for PCIDs other than the current one. Any one of the following suffices:
  - — Execute MOV to CR4 to modify CR4.PGE, either immediately or before again using any of the affected PCIDs. For example, software could use different (previously unused) PCIDs for the processes that used the affected PCIDs.
  - — For each affected PCID, execute MOV to CR3 to make that PCID current (and to load the address of the appropriate PML4 table). If the modified entry controls no global pages and bit 63 of the source operand to MOV to CR3 was 0, no further steps are required. Otherwise, execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry; if no page numbers that would use the entry have translations, execute INVLPG at least once.
- If software using PAE paging modifies a PDPTE, it should reload CR3 with the register's current value to ensure that the modified PDPTE is loaded into the corresponding PDPTE register (see Section 4.4.1).
- If the nature of the paging structures is such that a single entry may be used for multiple purposes (see Section 4.10.3.3), software should perform invalidations for all of these purposes. For example, if a single entry might serve as both a PDE and PTE, it may be necessary to execute INVLPG with two (or more) linear addresses, one that uses the entry as a PDE and one that uses it as a PTE. (Alternatively, software could use MOV to CR3 or MOV to CR4.)
- As noted in Section 4.10.2, the TLBs may subsequently contain multiple translations for the address range if software modifies the paging structures so that the page size used for a 4-KByte range of linear addresses changes. A reference to a linear address in the address range may use any of these translations.

  Software wishing to prevent this uncertainty should not write to a paging-structure entry in a way that would change, for any linear address, both the page size and either the page frame, access rights, or other attributes. It can instead use the following algorithm: first clear the P flag in the relevant paging-structure entry (e.g.,

1. One execution of INVLPG is sufficient even for a page with size greater than 4 KBytes.

PDE); then invalidate any translations for the affected linear addresses (see above); and then modify the relevant paging-structure entry to set the P flag and establish modified translation(s) for the new page size.

- Software should clear bit 63 of the source operand to a MOV to CR3 instruction that establishes a PCID that had been used earlier for a different linear-address space (e.g., with a different value in bits 51:12 of CR3). This ensures invalidation of any information that may have been cached for the previous linear-address space.

  This assumes that both linear-address spaces use the same global pages and that it is thus not necessary to invalidate any global TLB entries. If that is not the case, software should invalidate those entries by executing MOV to CR4 to modify CR4.PGE.

4.10.4.3 Optional Invalidation

The following items describe cases in which software may choose not to invalidate and the potential consequences of that choice:

- If a paging-structure entry is modified to change the P flag from 0 to 1, no invalidation is necessary. This is because no TLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the P flag is 0.[1]
- If a paging-structure entry is modified to change the accessed flag from 0 to 1, no invalidation is necessary (assuming that an invalidation was performed the last time the accessed flag was changed from 1 to 0). This is because no TLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the accessed flag is 0.
- If a paging-structure entry is modified to change the R/W flag from 0 to 1, failure to perform an invalidation may result in a "spurious" page-fault exception (e.g., in response to an attempted write access) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 4.10.4.1).
- If CR4.SMEP = 0 and a paging-structure entry is modified to change the U/S flag from 0 to 1, failure to perform an invalidation may result in a "spurious" page-fault exception (e.g., in response to an attempted user-mode access) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 4.10.4.1).
- If a paging-structure entry is modified to change the XD flag from 1 to 0, failure to perform an invalidation may result in a "spurious" page-fault exception (e.g., in response to an attempted instruction fetch) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 4.10.4.1).
- If a paging-structure entry is modified to change the accessed flag from 1 to 0, failure to perform an invalidation may result in the processor not setting that bit in response to a subsequent access to a linear address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such an access has not occurred.
- If software modifies a paging-structure entry that identifies the final physical address for a linear address (either a PTE or a paging-structure entry in which the PS flag is 1) to change the dirty flag from 1 to 0, failure to perform an invalidation may result in the processor not setting that bit in response to a subsequent write to a linear address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such a write has not occurred.
- The read of a paging-structure entry in translating an address being used to fetch an instruction may appear to execute before an earlier write to that paging-structure entry if there is no serializing instruction between the write and the instruction fetch. Note that the invalidating instructions identified in Section 4.10.4.1 are all serializing instructions.
- Section 4.10.3.3 describes situations in which a single paging-structure entry may contain information cached in multiple entries in the paging-structure caches. Because all entries in these caches are invalidated by any execution of INVLPG, it is not necessary to follow the modification of such a paging-structure entry by executing INVLPG multiple times solely for the purpose of invalidating these multiple cached entries. (It may be necessary to do so to invalidate multiple TLB entries.)

1. If it is also the case that no invalidation was performed the last time the P flag was changed from 1 to 0, the processor may use a TLB entry or paging-structure cache entry that was created when the P flag had earlier been 1.

4.10.4.4 Delayed Invalidation

Required invalidations may be delayed under some circumstances. Software developers should understand that, between the modification of a paging-structure entry and execution of the invalidation instruction recommended in Section 4.10.4.2, the processor may use translations based on either the old value or the new value of the paging-structure entry. The following items describe some of the potential consequences of delayed invalidation:

- If a paging-structure entry is modified to change the P flag from 1 to 0, an access to a linear address whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the R/W flag from 0 to 1, write accesses to linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the U/S flag from 0 to 1, user-mode accesses to linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the XD flag from 1 to 0, instruction fetches from linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.

As noted in Section 8.1.1, 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 and invalidation has been delayed, some of the accesses may complete (writing to memory) while another causes a page-fault exception.[1] In this case, the effects of the completed accesses may be visible to software even though the overall instruction caused a fault.

In some cases, the consequences of delayed invalidation may not affect software adversely. For example, when freeing a portion of the linear-address space (by marking paging-structure entries "not present"), invalidation using INVLPG may be delayed if software does not re-allocate that portion of the linear-address space or the memory that had been associated with it. However, because of speculative execution (or errant software), there may be accesses to the freed portion of the linear-address space before the invalidations occur. In this case, the following can happen:

- Reads can occur to the freed portion of the linear-address space. Therefore, invalidation should not be delayed for an address range that has read side effects.
- The processor may retain entries in the TLBs and paging-structure caches for an extended period of time. Software should not assume that the processor will not use entries associated with a linear address simply because time has passed.
- As noted in Section 4.10.3.1, the processor may create an entry in a paging-structure cache even if there are no translations for any linear address that might use that entry. Thus, if software has marked "not present" all entries in a page table, the processor may subsequently create a PDE-cache entry for the PDE that references that page table (assuming that the PDE itself is marked "present").
- If software attempts to write to the freed portion of the linear-address space, the processor might not generate a page fault. (Such an attempt would likely be the result of a software error.) For that reason, the page frames previously associated with the freed portion of the linear-address space should not be reallocated for another purpose until the appropriate invalidations have been performed.

4.10.5 Propagation of Paging-Structure Changes to Multiple Processors

As noted in Section 4.10.4, software that modifies a paging-structure entry may need to invalidate entries in the TLBs and paging-structure caches that were derived from the modified entry before it was modified. In a system containing more than one logical processor, software must account for the fact that there may be entries in the TLBs and paging-structure caches of logical processors other than the one used to modify the paging-structure entry. The process of propagating the changes to a paging-structure entry is commonly referred to as "TLB shootdown."

TLB shootdown can be done using memory-based semaphores and/or interprocessor interrupts (IPI). The following items describe a simple but inefficient example of a TLB shootdown algorithm for processors supporting the Intel-64 and IA-32 architectures:

1. If the accesses are to different pages, this may occur even if invalidation has not been delayed.

1. Begin barrier: Stop all but one logical processor; that is, cause all but one to execute the HLT instruction or to enter a spin loop.
2. Allow the active logical processor to change the necessary paging-structure entries.
3. Allow all logical processors to perform invalidations appropriate to the modifications to the paging-structure entries.
4. Allow all logical processors to resume normal operation.

Alternative, performance-optimized, TLB shootdown algorithms may be developed; however, software developers must take care to ensure that the following conditions are met:

- All logical processors that are using the paging structures that are being modified must participate and perform appropriate invalidations after the modifications are made.
- If the modifications to the paging-structure entries are made before the barrier or if there is no barrier, the operating system must ensure one of the following: (1) that the affected linear-address range is not used between the time of modification and the time of invalidation; or (2) that it is prepared to deal with the consequences of the affected linear-address range being used during that period. For example, if the operating system does not allow pages being freed to be reallocated for another purpose until after the required invalidations, writes to those pages by errant software will not unexpectedly modify memory that is in use.
- Software must be prepared to deal with reads, instruction fetches, and prefetch requests to the affected linear-address range that are a result of speculative execution that would never actually occur in the executed code path.

When multiple logical processors are using the same linear-address space at the same time, they must coordinate before any request to modify the paging-structure entries that control that linear-address space. In these cases, the barrier in the TLB shootdown routine may not be required. For example, when freeing a range of linear addresses, some other mechanism can assure no logical processor is using that range before the request to free it is made. In this case, a logical processor freeing the range can clear the P flags in the PTEs associated with the range, free the physical page frames associated with the range, and then signal the other logical processors using that linear-address space to perform the necessary invalidations. All the affected logical processors must complete their invalidations before the linear-address range and the physical page frames previously associated with that range can be reallocated.

4.11 INTERACTIONS WITH VIRTUAL-MACHINE EXTENSIONS (VMX)

The architecture for virtual-machine extensions (VMX) includes features that interact with paging. Section 4.11.1 discusses ways in which VMX-specific control transfers, called VMX transitions specially affect paging. Section 4.11.2 gives an overview of VMX features specifically designed to support address translation.

4.11.1 VMX Transitions

The VMX architecture defines two control transfers called **VM entries** and **VM exits**; collectively, these are called **VMX transitions**. VM entries and VM exits are described in detail in Chapter 26 and Chapter 27, respectively, in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*. The following items identify paging-related details:

- VMX transitions modify the CR0 and CR4 registers and the IA32_EFER MSR concurrently. For this reason, they allow transitions between paging modes that would not otherwise be possible:
  - — VM entries allow transitions from 4-level paging directly to either 32-bit paging or PAE paging.
  - — VM exits allow transitions from either 32-bit paging or PAE paging directly to 4-level paging or 5-level paging.
- VMX transitions that result in PAE paging load the PDPTE registers (see Section 4.4.1) as follows:
  - — VM entries load the PDPTE registers either from the physical address being loaded into CR3 or from the virtual-machine control structure (VMCS); see Section 26.3.2.4.
  - — VM exits load the PDPTE registers from the physical address being loaded into CR3; see Section 27.5.4.

- VMX transitions invalidate the TLBs and paging-structure caches based on certain control settings. See Section 26.3.2.5 and Section 27.5.5 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.

4.11.2 VMX Support for Address Translation

Chapter 28, "VMX Support for Address Translation," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* describe two features of the virtual-machine extensions (VMX) that interact directly with paging. These are **virtual-processor identifiers** (**VPIDs**) and the **extended page table** mechanism (**EPT**).

VPIDs provide a way for software to identify to the processor the address spaces for different "virtual processors." The processor may use this identification to maintain concurrently information for multiple address spaces in its TLBs and paging-structure caches, even when non-zero PCIDs are not being used. See Section 28.1 for details.

When EPT is in use, the addresses in the paging-structures are not used as physical addresses to access memory and memory-mapped I/O. Instead, they are treated as **guest-physical** addresses and are translated through a set of EPT paging structures to produce physical addresses. EPT can also specify its own access rights and memory typing; these are used on conjunction with those specified in this chapter. See Section 28.3 for more information.

Both VPIDs and EPT may change the way that a processor maintains information in TLBs and paging structure caches and the ways in which software can manage that information. Some of the behaviors documented in Section 4.10 may change. See Section 28.4 for details.

4.12 USING PAGING FOR VIRTUAL MEMORY

With paging, portions of the linear-address space need not be mapped to the physical-address space; data for the unmapped addresses can be stored externally (e.g., on disk). This method of mapping the linear-address space is referred to as virtual memory or demand-paged virtual memory.

Paging divides the linear address space into fixed-size pages that can be mapped into the physical-address space and/or external storage. When a program (or task) references a linear address, the processor uses paging to translate the linear address into a corresponding physical address if such an address is defined.

If the page containing the linear address is not currently mapped into the physical-address space, the processor generates a page-fault exception as described in Section 4.7. The handler for page-fault exceptions typically directs the operating system or executive to load data for the unmapped page from external storage into physical memory (perhaps writing a different page from physical memory out to external storage in the process) and to map it using paging (by updating the paging structures). When the page has been loaded into physical memory, a return from the exception handler causes the instruction that generated the exception to be restarted.

Paging differs from segmentation through its use of fixed-size pages. Unlike segments, which usually are the same size as the code or data structures they hold, pages have a fixed size. If segmentation is the only form of address translation used, a data structure present in physical memory will have all of its parts in memory. If paging is used, a data structure can be partly in memory and partly in disk storage.

4.13 MAPPING SEGMENTS TO PAGES

The segmentation and paging mechanisms provide support for a wide variety of approaches to memory management. When segmentation and paging are combined, segments can be mapped to pages in several ways. To implement a flat (unsegmented) addressing environment, for example, all the code, data, and stack modules can be mapped to one or more large segments (up to 4-GBytes) that share same range of linear addresses (see Figure 3-2 in Section 3.2.2). Here, segments are essentially invisible to applications and the operating-system or executive. If paging is used, the paging mechanism can map a single linear-address space (contained in a single segment) into virtual memory. Alternatively, each program (or task) can have its own large linear-address space (contained in its own segment), which is mapped into virtual memory through its own paging structures.

Segments can be smaller than the size of a page. If one of these segments is placed in a page which is not shared with another segment, the extra memory is wasted. For example, a small data structure, such as a 1-Byte sema-

phore, occupies 4 KBytes if it is placed in a page by itself. If many semaphores are used, it is more efficient to pack them into a single page.

The Intel-64 and IA-32 architectures do not enforce correspondence between the boundaries of pages and segments. A page can contain the end of one segment and the beginning of another. Similarly, a segment can contain the end of one page and the beginning of another.

Memory-management software may be simpler and more efficient if it enforces some alignment between page and segment boundaries. For example, if a segment which can fit in one page is placed in two pages, there may be twice as much paging overhead to support access to that segment.

One approach to combining paging and segmentation that simplifies memory-management software is to give each segment its own page table, as shown in Figure 4-13. This convention gives the segment a single entry in the page directory, and this entry provides the access control information for paging the entire segment.

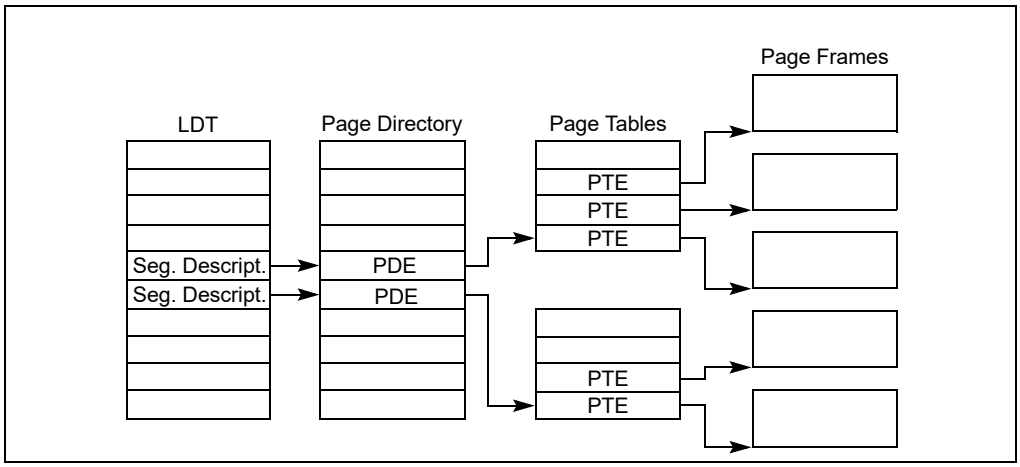


Figure 4-13. Memory Management Convention That Assigns a Page Table to Each Segment

12. Updates to Chapter 6, Volume 3A

Changes to Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume3A:* System Programming Guide, Part 1.

Changes to this chapter: HLAT updates made to Interrupt 14—Page-Fault Exception (#PF).

CHAPTER 6
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 20, "8086 Emulation," describes information specific to interrupt and exception mechanisms in real-address and virtual-8086 mode. Section 6.14, "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.

6.1 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.

6.2 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 6.10, "Interrupt Descriptor Table (IDT)").

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 6.3, "Sources of Interrupts").

Table 6-1 shows vector number assignments for architecturally defined exceptions and for the NMI interrupt. This table gives the exception type (see Section 6.5, "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.

6.3 SOURCES OF INTERRUPTS

The processor receives interrupts from two sources:

- External (hardware generated) interrupts.
- Software-generated interrupts.

6.3.1 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 10, "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 6.2, "Exception and Interrupt Vectors"). Asserting the NMI pin signals a non-maskable interrupt (NMI), which is assigned to interrupt vector 2.

Table 6-1. 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 6-1. 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 (Zero)	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 *n* instruction.

NOTES:

1. 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.

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 31, "System Management Mode."

6.3.2 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 6.8.1, "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.

6.3.3 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.

6.4 SOURCES OF EXCEPTIONS

The processor receives exceptions from three sources:

- Processor-detected program-error exceptions.
- Software-generated exceptions.
- Machine-check exceptions.

6.4.1 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 6.5, "Exception Classifications").

6.4.2 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.

6.4.3 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 depen-

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.

dent. When a machine-check error is detected, the processor signals a machine-check exception (vector 18) and returns an error code.

See Chapter 6, "Interrupt 18—Machine-Check Exception (#MC)" and Chapter 15, "Machine-Check Architecture," for more information about the machine-check mechanism.

6.5 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.

6.6 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).

6.7 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 6.7.1, "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.

6.7.1 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 6.8.1, "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 20.2.7, "Sensitive Instructions"). In such a case, NMIs are unmasked before the exception handler is invoked.

6.8 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.

6.8.1 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 6.3.2, "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 6.4.2, "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 and IRET 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.)

See the descriptions of the CLI, STI, PUSHF, POPF, and IRET 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.

6.8.2 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 17.3.1.1, "Instruction-Breakpoint Exception Condition," for more information on the use of this flag.

As noted in Section 6.8.3, 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).

---

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 20.3, "Interrupt and Exception Handling in Virtual-8086 Mode." Behavior is also impacted by the PVI flag: see Section 20.4, "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.

6.8.3 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.

6.9 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 6-2 shows the priority among classes of event sources.

Table 6-2. 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]
7	Fault-class Debug Exceptions (#DB due to instruction breakpoint)

Table 6-2. Priority Among Concurrent Events (Contd.)

Priority	Description
8	Faults from Fetching Next Instruction - Code-Segment Limit Violation (#GP) - Code Page Fault (#PF) - Control protection exception due to missing ENDBRANCH at target of an indirect call or jump (#CP)
9 (Lowest)	Faults from Decoding the Next Instruction - 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 6-2 is consistent across processor implementations, the priority of events within a class is implementation-dependent and may vary from processor to processor.

Table 6-2 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.

6.10 INTERRUPT DESCRIPTOR TABLE (IDT)

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 6-1, 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.

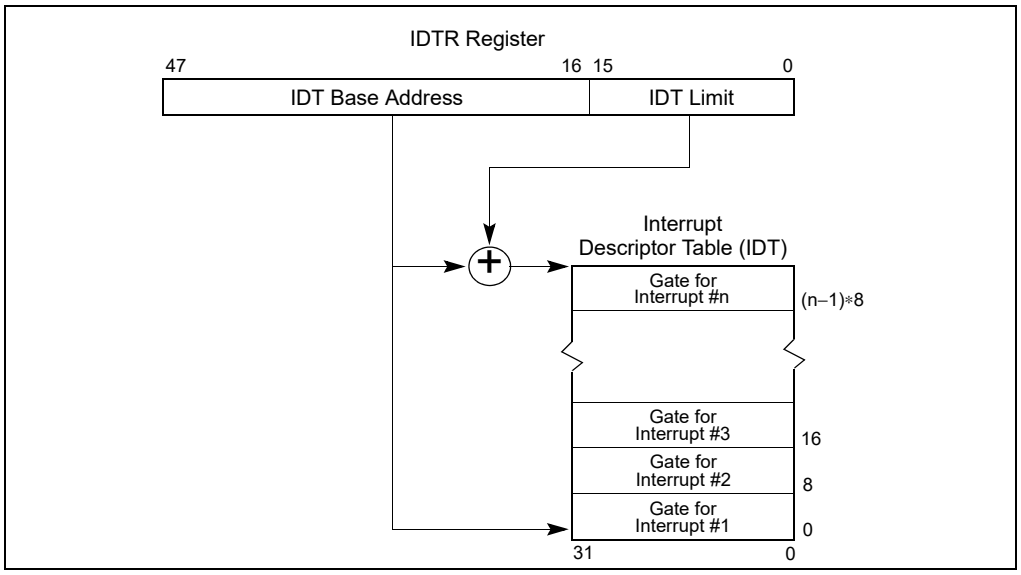


Figure 6-1. Relationship of the IDTR and IDT

6.11 IDT DESCRIPTORS

The IDT may contain any of three kinds of gate descriptors:

- Task-gate descriptor
- Interrupt-gate descriptor
- Trap-gate descriptor

Figure 6-2 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 7.2.5, "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 5.8.3, "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 6.12.1.3, "Flag Usage By Exception- or Interrupt-Handler Procedure").

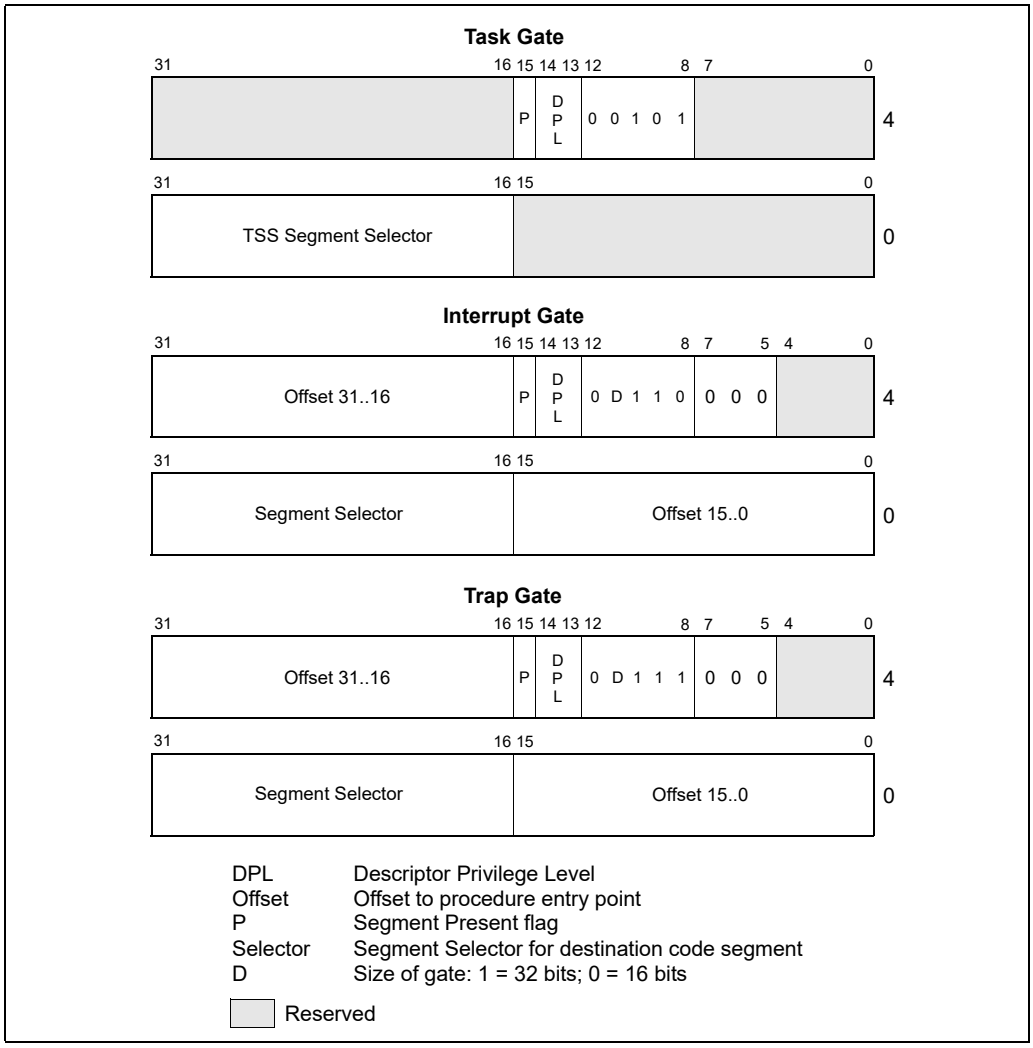


Figure 6-2. IDT Gate Descriptors

6.12 EXCEPTION AND INTERRUPT HANDLING

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 5.8.2, “Gate Descriptors,” through Section 5.8.6, “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 7.3, “Task Switching”).

6.12.1 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 6-3). 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.

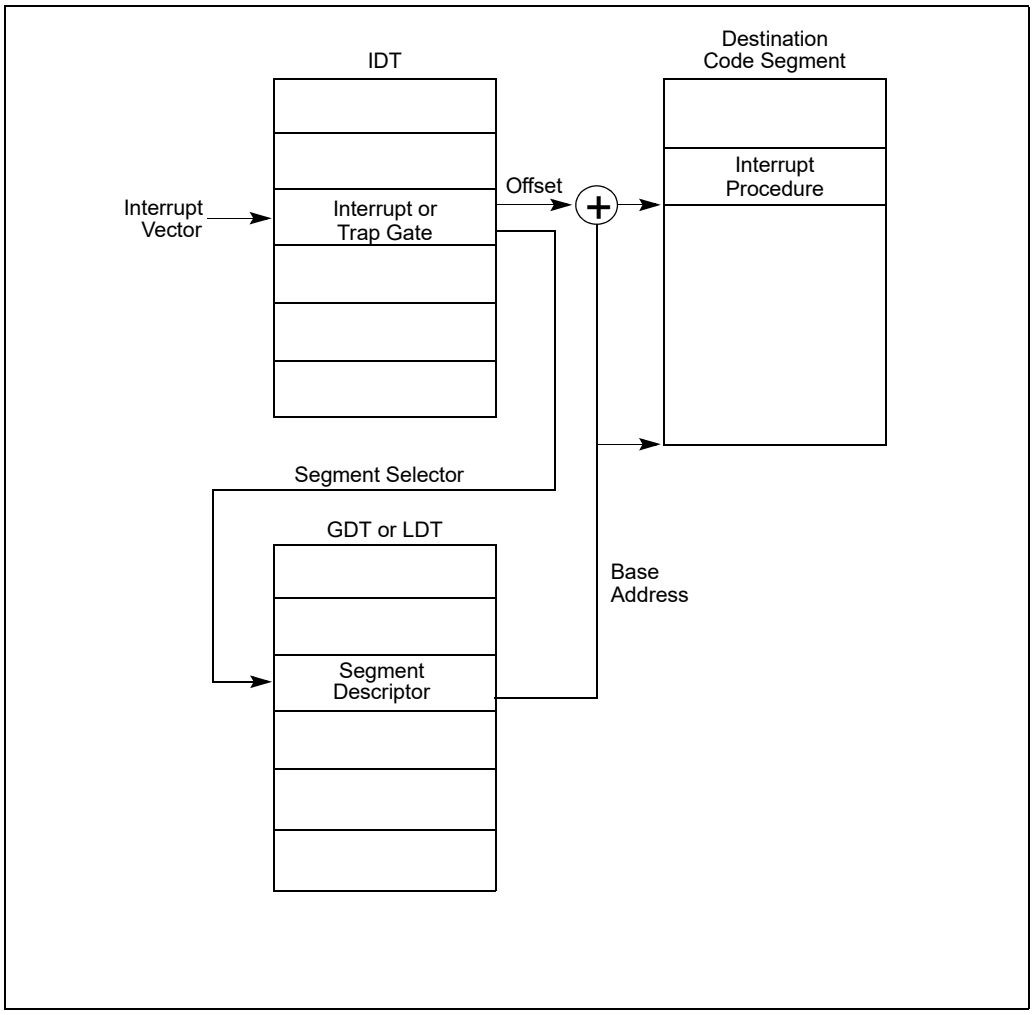


Figure 6-3. Interrupt Procedure Call

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 6-4).
  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 6-4).
  b. If an exception causes an error code to be saved, it is pushed on the current stack after the EIP value.

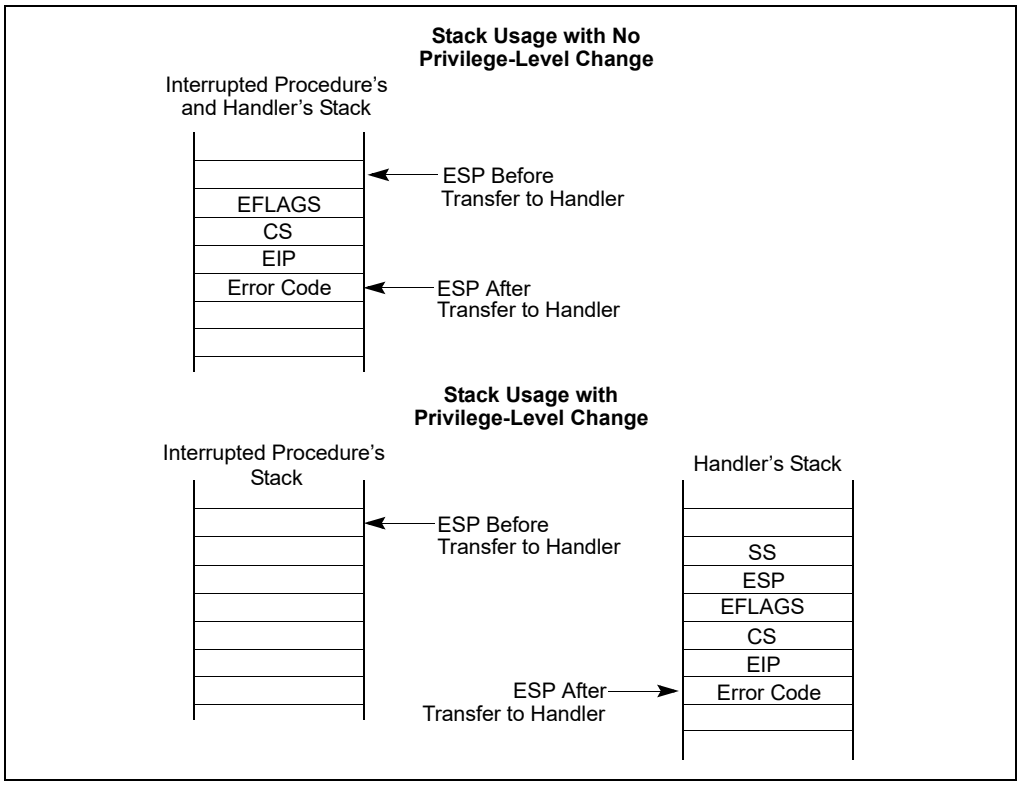


Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

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.

6.12.1.1 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 execute 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 6-5.
- 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 6-5.

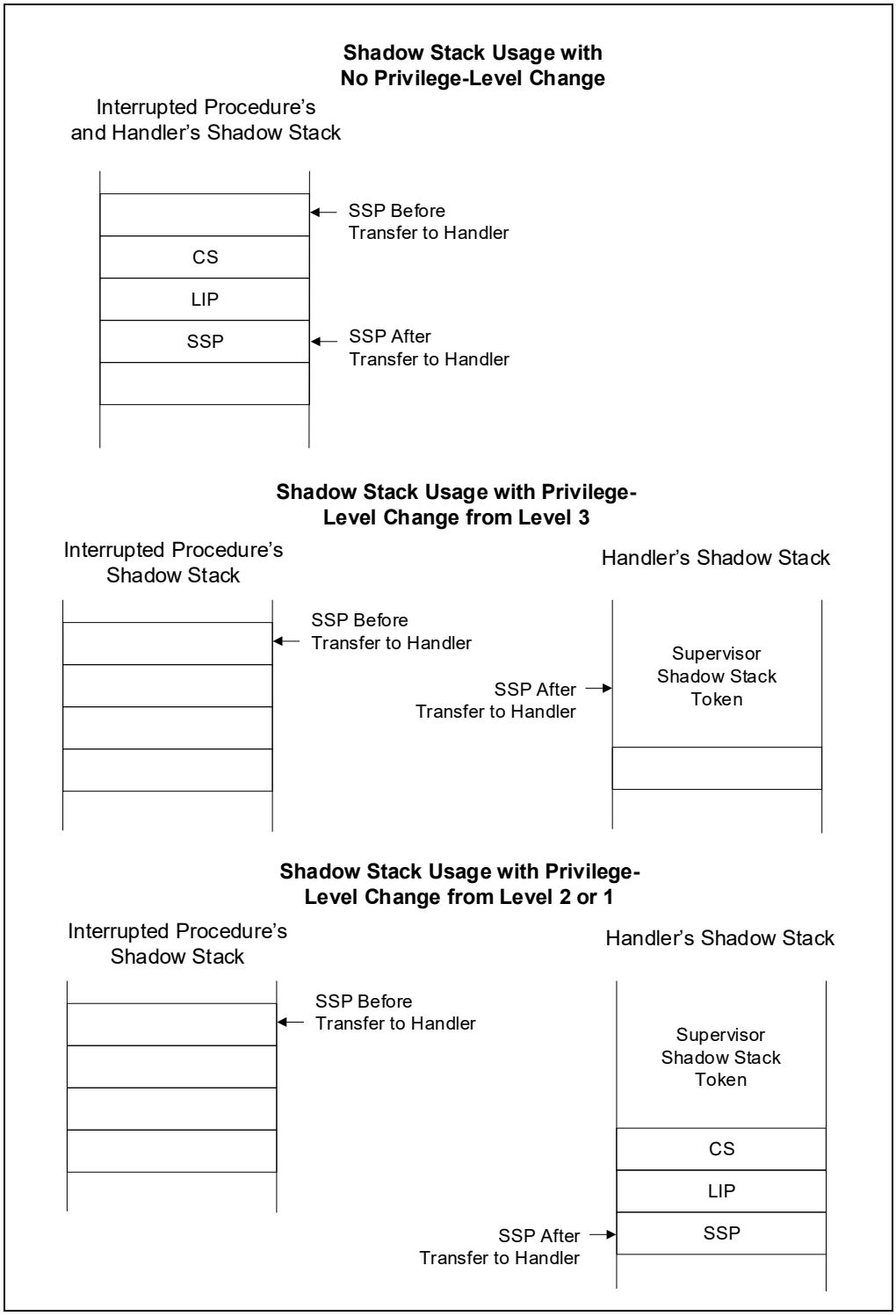


Figure 6-5. Shadow Stack Usage on Transfers to Interrupt and Exception-Handling Routines

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.

6.12.1.2 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 5.8.4, "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).

6.12.1.3 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.

6.12.2 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 6-6). A switch to the handler task is handled in the same manner as an ordinary task switch (see Section 7.3, “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.

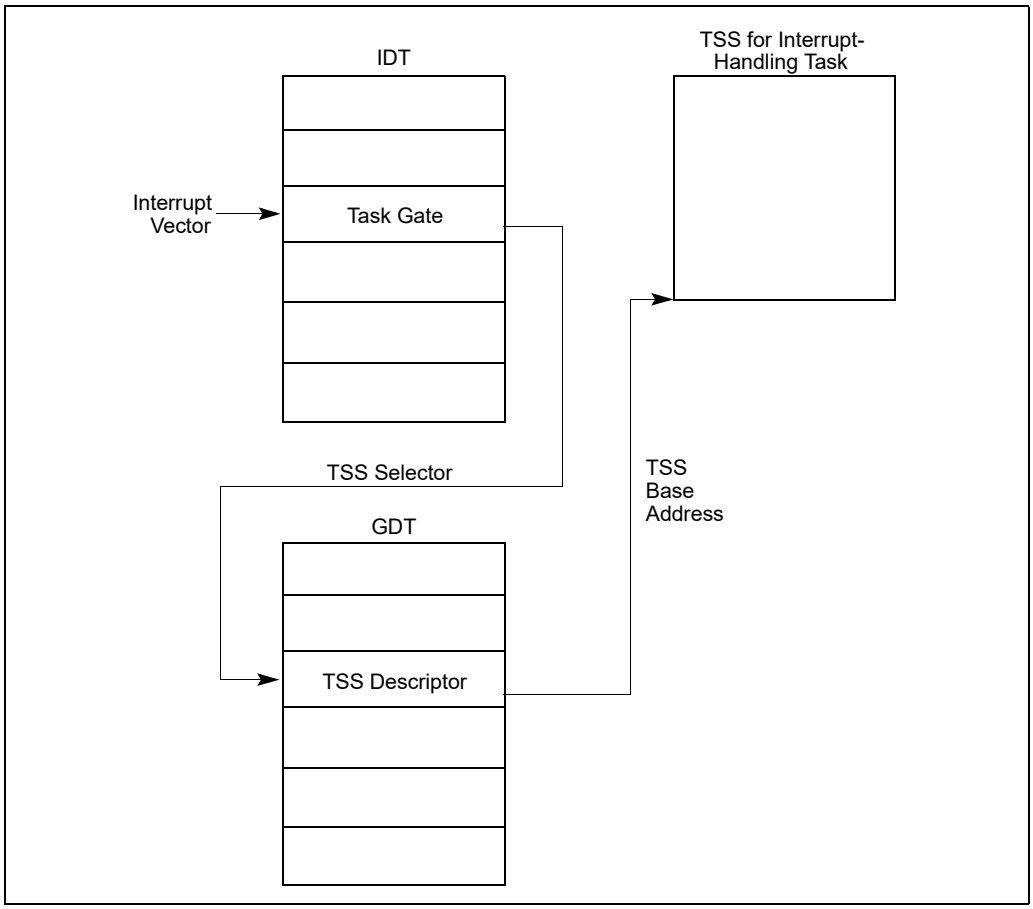


Figure 6-6. Interrupt Task Switch

6.13 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 6-7. 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.

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.

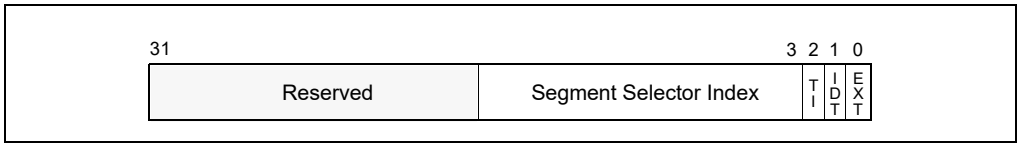


Figure 6-7. Error Code

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.

6.14 EXCEPTION AND INTERRUPT HANDLING IN 64-BIT MODE

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.

6.14.1 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 6-8.

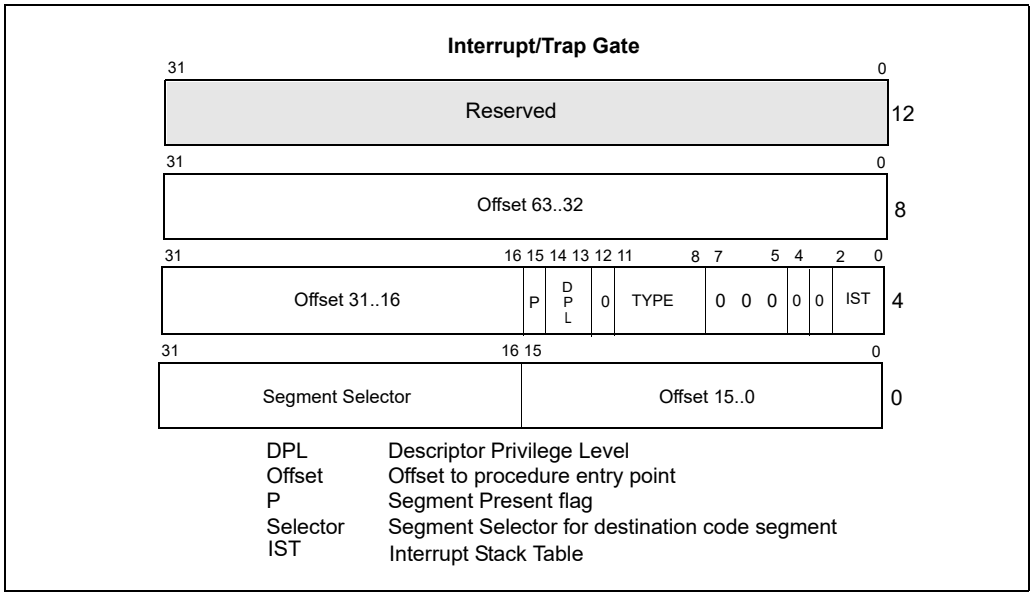


Figure 6-8. 64-Bit IDT Gate Descriptors

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 6.14.5, "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.

6.14.2 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.

6.14.3 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 ≠ 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.

6.14.4 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 6-9). 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.

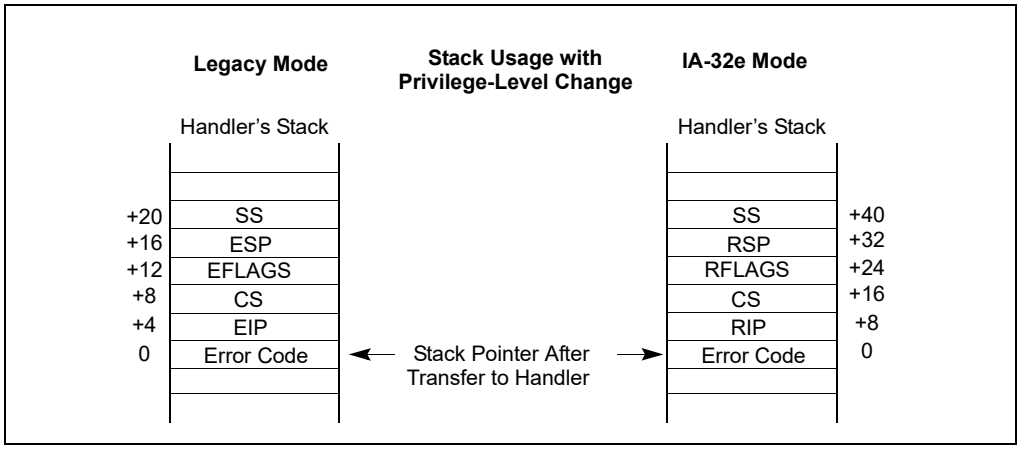


Figure 6-9. IA-32e Mode Stack Usage After Privilege Level Change

6.14.5 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 6-8. 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, 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, the processor requires that the shadow stack addresses programmed into this table point to a supervisor shadow stack token; see Figure 6-10.

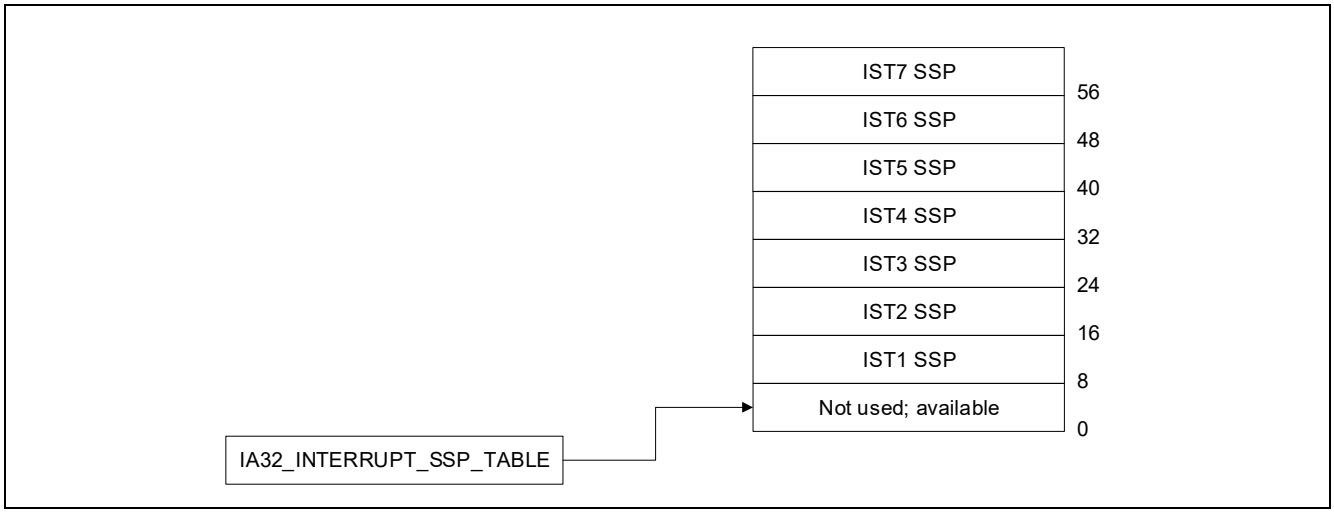


Figure 6-10. Interrupt Shadow Stack Table

6.15 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 6-3). See Chapter 17, "Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features," for detailed information about the debug exceptions.

Table 6-3. 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 16.3.7, "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 6.5, "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 17.3.2, "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 16.3.7, "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/INTO/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, PREFETCH*h*, 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, PREFETCH*h*, SFENCE, LFENCE, MFENCE, and CLFLUSH; or the 64-bit versions of the PAVGB, PAVGW, PEXTRW, PINSRW, PMAXSW, PMAXUB, PMINSW, PMINUB, PMOVMSKB, PMULHUW, PSADBW, PSHUFW, PADDQ, PSUBQ, PALIGNR, PABSB, PABSD, PABSW, PHADDD, PHADDSW, PHADDW, PHSUBD, PHSUBSW, PHSUBW, PMADDUBSM, PMULHRSW, 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 or UD2 instruction. Note that even though it is the execution of the UD0, UD1 or UD2 instruction that causes the invalid opcode exception, the saved instruction pointer will still points at the UD0, UD1 or UD2 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 6.5, "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, PREFETCH*h*, 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 6-4).

Table 6-4. Interrupt and Exception Classes

Class	Vector Number	Description
Benign Exceptions and Interrupts	1 2 3 4 5 6 7 9 16 17 18 19 All All	Debug NMI Interrupt Breakpoint Overflow BOUND Range Exceeded Invalid Opcode Device Not Available Coprocessor Segment Overrun Floating-Point Error Alignment Check Machine Check SIMD floating-point INT *n* INTR
Contributory Exceptions	0 10 11 12 13 21	Divide Error Invalid TSS Segment Not Present Stack Fault General Protection Control Protection
Page Faults	14 20	Page Fault Virtualization Exception

Table 6-5 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 6-5. 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 6-5. 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—Coprocessor 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 6-6 shows the conditions that cause an invalid TSS exception to be generated.

Table 6-6. 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 ≠ CPL.
Stack segment selector index	The stack segment selector RPL ≠ 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 ≠ 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 6-6. 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 generated either in the context of the original task or in the context of the new task (see Section 7.3, "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 than causes the fault. See Section 7.3, "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 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 7.3, “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 7.3, “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 5, "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 5.9, "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 7.3, “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 4, "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 Chapter 4.6, "Access Rights."
- An enclave access violates one of the specified access-control requirements. See Section 34.3, "Access-control Requirements" and Section 34.20, "Enclave Page Cache Map (EPCM)" in Chapter 34, "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 4.7, "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 6-11). 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 4.6, “Access Rights” and Section 4.7, “Page-Fault Exceptions” for more information about page-fault exceptions and the error codes that they produce.

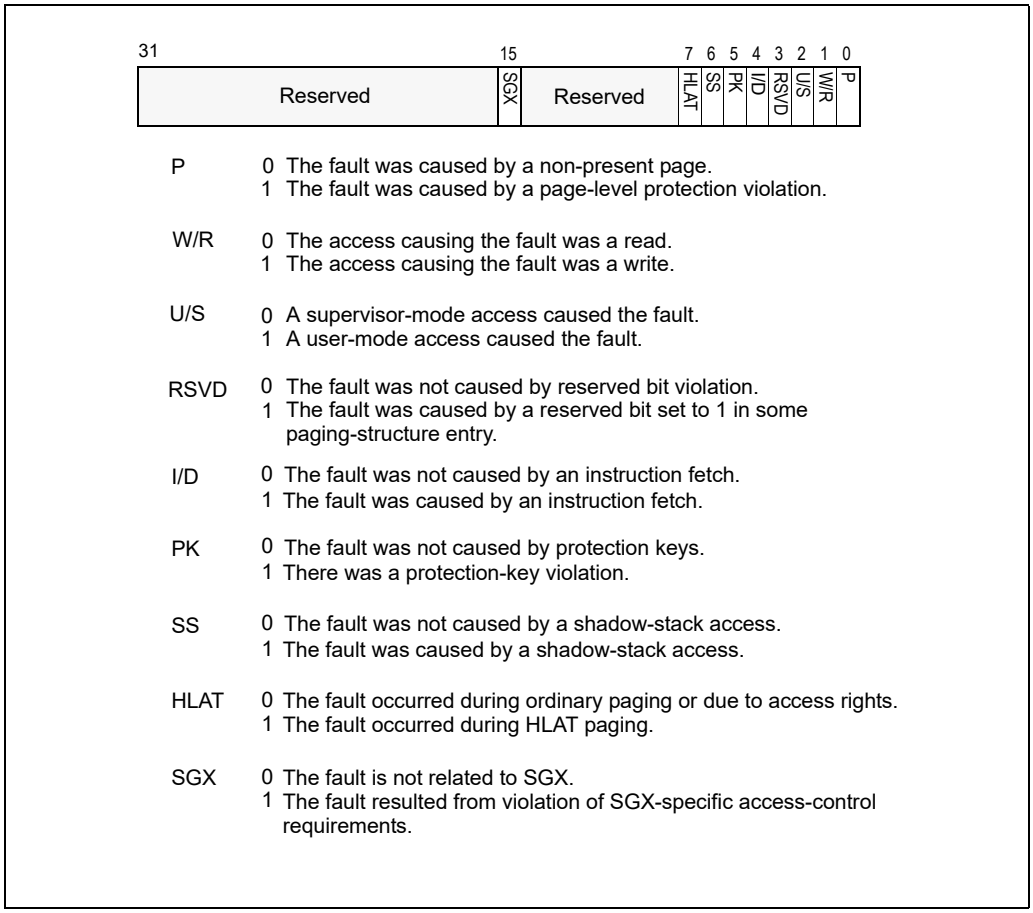


Figure 6-11. Page-Fault Error Code

- The contents of the CR2 register. The processor loads the CR2 register with the 32-bit linear address that generated the exception. The page-fault handler can use this address to locate the corresponding page directory and page-table 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 access flag in the page-directory entry is set when the fault occurs. The behavior of IA-32 processors regarding the access flag in the corresponding page-table entry 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).

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.

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.

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

Indicates that the processor detected 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-check 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 6-7 lists the alignment requirements various data types recognized by the processor.

Table 6-7. 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 the alignment check exception (#AC) is generated 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-check exceptions (#AC) 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-check exceptions, 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-check exception. 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 the alignment-check exception (#AC) is enabled when executing these instructions (and CPL is 3), a misaligned memory operand can cause either an alignment-check exception 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 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C*).

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-check exceptions (#AC) 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-check faults. These instructions are rarely needed by application programs.

Exception Error Code

Yes. The error code is null; all bits are clear except possibly bit 0 — EXT; see Section 6.13. EXT is set if the #AC is recognized during delivery of an event other than a software interrupt (see "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)*.

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.

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 15, "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 15.3.1.2, "IA32_MCG_STATUS MSR," and Section 15.3.2.6, "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 15.3.1.2, "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 OSXMMEXCPT 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 6-8. This exception precedence sometimes results in the higher priority exception condition being reported and the lower priority exception conditions being ignored.

Table 6-8. 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 25.5.7.2 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 25.5.7.2 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.

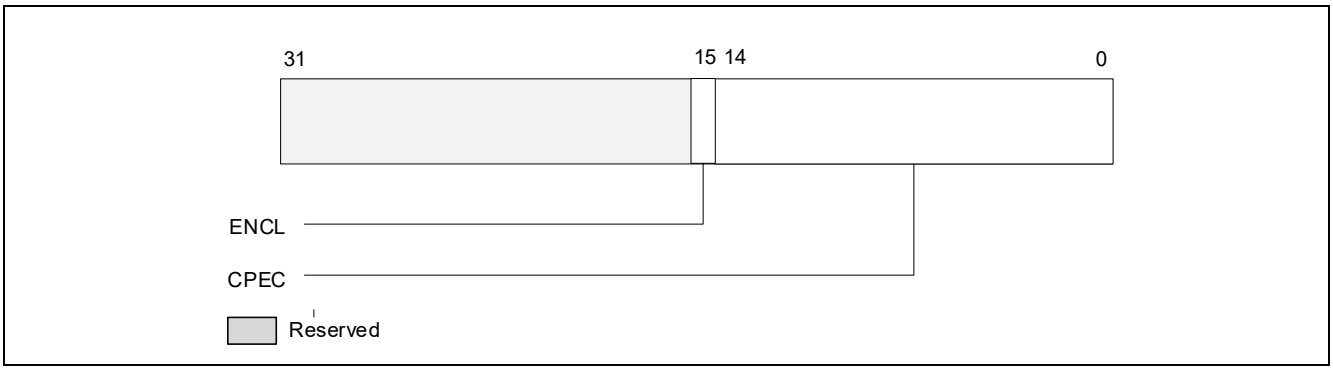


Figure 6-12. Exception Error Code Information

- 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.

13. Updates to Chapter 8, Volume 3A

Change bars and green 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: Update to Section 8.1.1, "Guaranteed Atomic Operations" and update to Section 8.2.5, "Strengthening or Weakening the Memory-Ordering Model".

CHAPTER 8
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 10, "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 8.5, "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 11. The APIC architecture is described in Chapter 10. Bus and memory locking, serializing instructions, memory ordering, and Intel Hyper-Threading Technology are discussed in the following sections.

8.1 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.

8.1.1 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[bit 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 4.10.4.4), such page faults may occur even if all accesses are to the same page.

8.1.2 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. 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 8.1.4, "Effects of a LOCK Operation on Internal Processor Caches").

8.1.2.1 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 setting the B (busy) flag of a TSS descriptor** — The processor tests and sets the busy flag in the type field of the TSS descriptor when switching to a task. 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 updating segment descriptors** — When loading a segment descriptor, the processor will set 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.
- **When updating page-directory and page-table entries** — When updating page-directory and page-table entries, the processor uses locked cycles to set the accessed and dirty flag in the page-directory and page-table entries.
- **Acknowledging interrupts** — 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.

8.1.2.2 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, and CMPXCHG8B).
- 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 signalling 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 recommend 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.

8.1.3 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;
```

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.

8.1.4 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 it's 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.

8.2 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 8.2.1 and Section 8.2.2 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 8.2.3 gives examples illustrating the behavior of the memory-ordering model on IA-32 and Intel-64 processors. Section 8.2.4 considers the special treatment of stores for string operations and Section 8.2.5 discusses how memory-ordering behavior may be modified through the use of specific instructions.

8.2.1 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 8.2.5, "Strengthening or Weakening the Memory-Ordering Model").

8.2.2 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 8.2.4.1).
- 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.

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.

- 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.
- 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 8-1. 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 8.2.4, "Fast-String Operation and Out-of-Order Stores," below).

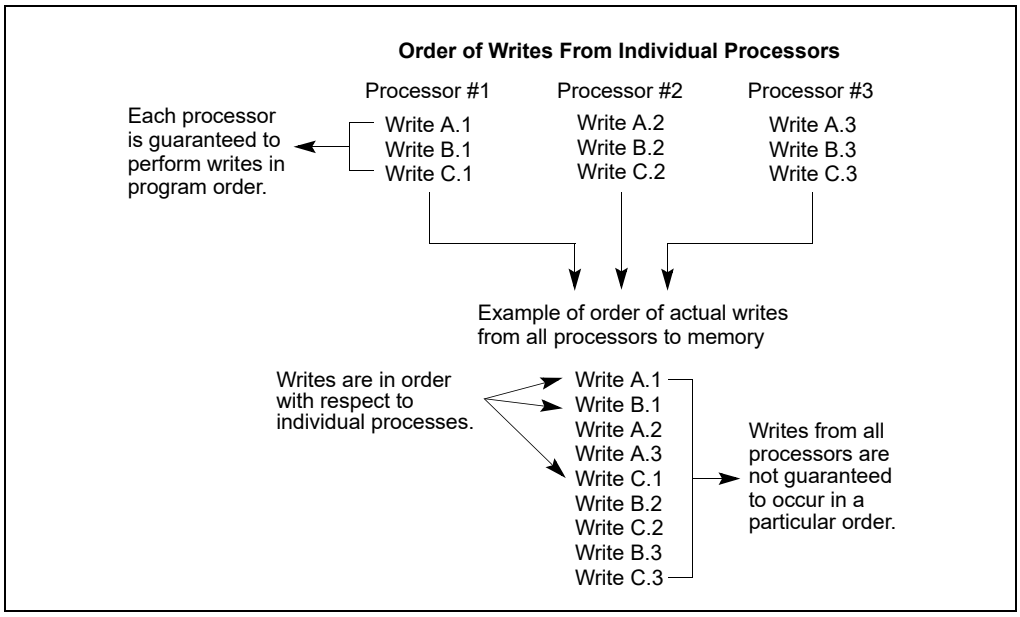


Figure 8-1. Example of Write Ordering in Multiple-Processor Systems

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.

8.2.3 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 8.2.2. 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 8.2.3.1 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.

8.2.3.1 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 8.2.4); 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 8.2.3.2 through Section 8.2.3.7 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 8.2.3.8 and Section 8.2.3.9 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" is 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."

8.2.3.2 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 8-1. Stores Are Not Reordered with Other Stores

Processor 0	Processor 1
mov [_x], 1 mov [_y], 1	mov r1, [_y] 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.

8.2.3.3 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 by the following example:

Example 8-2. 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.

8.2.3.4 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 8-3. 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 8-4. 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.

8.2.3.5 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 8-5. 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.

8.2.3.6 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 8-6. 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 8.2.3.3), 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 8.2.3.2), 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.

8.2.3.7 Stores Are Seen in a Consistent Order by Other Processors

As noted in Section 8.2.3.5, 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 8-7. 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 8.2.3.2,

- 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

8.2.3.8 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 8-8. Locked Instructions Have a Total Order

Processor 0	Processor 1	Processor 2	Processor 3
xchg [_x], r1	xchg [_y], r2		
		mov r3, [_x]	mov r5, [_y]
		mov r4, [_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 8.2.3.2), 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.

8.2.3.9 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 8-9. Loads Are not Reordered with Locks

Processor 0	Processor 1
xchg [_x], r1	xchg [_y], r3
mov r2, [_y]	mov r4, [_x]
Initially x = y = 0, r1 = r3 = 1 r2 = 0 and r4 = 0 is not allowed	

As explained in Section 8.2.3.8, 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 8-10. Stores Are not Reordered with Locks

Processor 0	Processor 1
xchg [_x], r1	mov r2, [_y]
mov [_y], 1	mov r3, [_x]

Example 8-10. Stores Are not Reordered with Locks

Processor 0	Processor 1
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 8.2.3.2), 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.

8.2.4 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. 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 8.2.4.1 and Section 8.2.4.2 provide further explain and examples.

8.2.4.1 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.
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 incorruptibility 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.

8.2.4.2 Examples Illustrating Memory-Ordering Principles for String Operations

The following examples uses the same notation and convention as described in Section 8.2.3.1.

In Example 8-11, 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 8-11. 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 <= _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 8-12, 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 8-12. 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 8-13, 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 8-13. String Operations Are not Reordered with later Stores

Processor 0	Processor 1
rep:stosd [_x]	mov r1, [_z]
mov [_z], $1	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 8-13 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 8-14 illustrates the visibility principle when a string operation is interrupted.

Example 8-14. Interrupted String Operation

Processor 0	Processor 1
rep:stosd [_x] // interrupted before es:edi reach _y	mov r1, [_z]
mov [_z], $1 // interrupt handler	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 8-14, 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 8-15 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 8-15. String Operations Are not Reordered with Earlier Stores

Processor 0	Processor 1
mov [_z], $1	mov r1, [_y]
rep:stosd [_x]	mov r2, [_z]
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	

8.2.5 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 11.11, "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 11.12, "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 8.1.2, "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 8.3). 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,

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.

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 11-7 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.

8.3 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 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 CR8[1]), MOV (to debug register), WBINVD, and WRMSR[2].
- **Non-privileged serializing instructions** — CPUID, IRET, and RSM.

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.

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.

The SFENCE, LFENCE, and MFENCE instructions provide more granularity in controlling the serialization of memory loads and stores (see Section 8.2.5, "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 4.10.4, "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.

8.4 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 summarizes the evolution of the changes:

- **For P6 family or older processors supporting MP operations**— The selection of the BSP and APs (see Section 8.4.1, "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 8.11.1, "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 8.4.1, "BSP and AP Processors") is handled through arbitration on the system bus, using BIPI and FIPI messages (see Section 8.4.3, "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 8.4.3, "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 8.4.3, "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 given in the EAX register when the CPUID instruction is executed with a value of 1 in the EAX register.

8.4.1 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 10-5) 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.

8.4.2 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.

8.4.3 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 leaf 0BH, otherwise the unique ID is an 8-bit value. (see Section 8.4.5, "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 8.4), 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 8.11.1, "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 8.4.4, "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.

8.4.4 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 10-1.

```
ICR_LOW         EQU 0FEE00300H
SVR             EQU 0FEE000F0H
APIC_ID         EQU 0FEE00020H
LVT3            EQU 0FEE00370H
APIC_ENABLED    EQU 0100H
BOOT_ID         DD ?
COUNT           EQU 00H
VACANT          EQU 00H
```

8.4.4.1 Typical BSP Initialization Sequence

After the BSP and APs have been selected (by means of a hardware protocol, see Section 8.4.3, "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 CPUID 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 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. 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, 0FF000000H; 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, FFFFFF00H; 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. 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 8-1 provides an algorithm when the expected processor count is unknown. The right-hand-side of Table 8-1 can be used when the expected processor count is known.

Table 8-1. 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.

8.4.4.2 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 8.4.4.1, "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.

8.4.5 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 (see *Intel® 64 Architecture x2APIC Specification*)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 leaf 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 10.4.6, "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 leaf 0BH) — An APIC ID is assigned to a logical processor during power up. This is the initial APIC ID reported by CPUID.1: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 leaf 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 level of the multi-threading resource topology in an MP system (See Section 8.9.1, "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 leaf 0BH** (If the processor supports CPUID leaf 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 leaf 0BH to determine the topological relationship between logical processors if the processor supports CPUID leaf 0BH.

  Bits in the 32-bit x2APIC ID can be extracted into sub-fields using CPUID leaf 0BH parameters. (See Section 8.9.1, "Hierarchical Mapping of Shared Resources").

Figure 8-2 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 8.9.3, "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 8.9.1, "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 8.9.

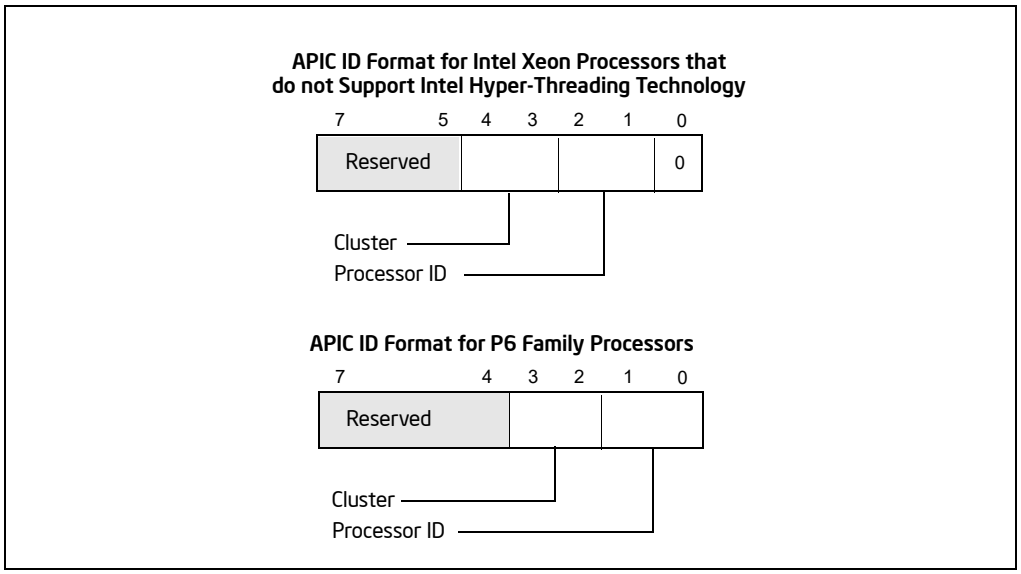


Figure 8-2. Interpretation of APIC ID in Early MP Systems

For P6 family processors, the APIC ID that is assigned to a processor during power-up and initialization is 4 bits (see Figure 8-2). Here, bits 0 and 1 form a 2-bit processor (or socket) identifier and bits 2 and 3 form a 2-bit cluster ID.

8.5 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 8.7, "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 8.6.2, "Initializing Multi-Core Processors").

8.6 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.1: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.1: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 Package[2] (CPUID.(EAX=4, ECX=0[3]):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 leaf 0BH will assign unique APIC IDs to each logical processor in the system. CPUID leaf 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 executing CPUID with EAX= 0. If CPUID.0H:EAX is greater than or equal or 11 (0BH), then proceed to next step,
- Check CPUID.EAX=0BH, ECX=0H:EBX is non-zero.

If both of the above conditions are true, extended topology enumeration leaf is available. Note the presence of CPUID leaf 0BH in a processor does not guarantee support that the local APIC supports x2APIC. If CPUID.(EAX=0BH, ECX=0H):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.

---

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.
2. Software must check CPUID for its support of leaf 4 when implementing support for multi-core. If CPUID leaf 4 is not available at runtime, software should handle the situation as if there is only one core per package.
3. 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.

8.6.1 Initializing Processors Supporting 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 8.4, "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 8.4.3, "MP Initialization Protocol Algorithm for MP Systems," and Section 8.4.4, "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 8.9.3, "Hierarchical ID of Logical Processors in an MP System"). Once logical processors have APIC IDs, software communicates with them by sending APIC IPI messages.

8.6.2 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 8.4, "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.

8.6.3 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.

8.6.4 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 8-3).

Logical processors can also send IPIs to other logical processors by writing to the ICR register of its local APIC (see Section 10.6, "Issuing Interprocessor Interrupts"). This also applies to dual-core processors.

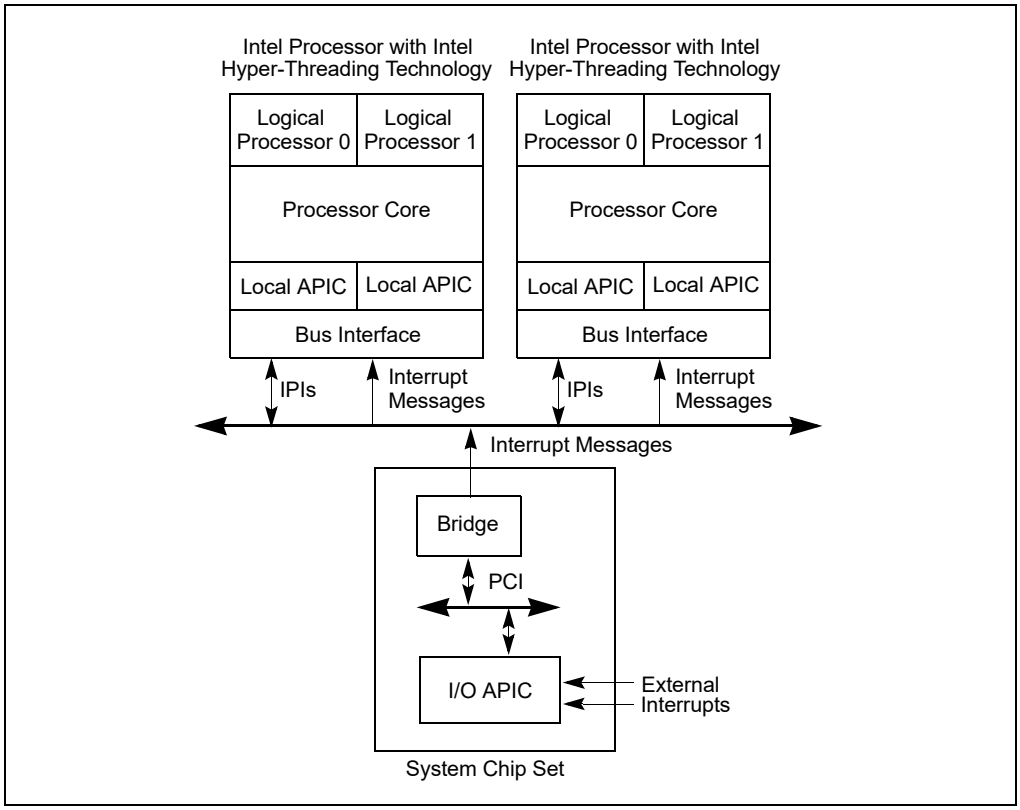


Figure 8-3. Local APICs and I/O APIC in MP System Supporting Intel HT Technology

8.7 INTEL® HYPER-THREADING TECHNOLOGY ARCHITECTURE

Figure 8-4 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).

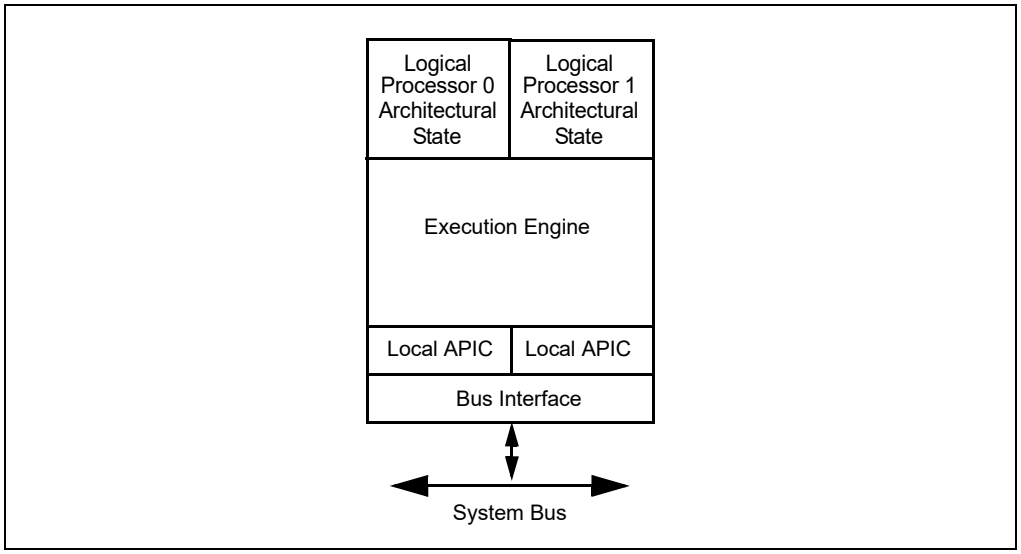


Figure 8-4. IA-32 Processor with Two Logical Processors Supporting Intel HT Technology

8.7.1 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

8.7.2 APIC Functionality

When a processor supporting Intel Hyper-Threading Technology support is initialized, each logical processor is assigned a local APIC ID (see Table 10-1). 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 8.9.3, "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 10, "Advanced Programmable Interrupt Controller (APIC)," for a detailed discussion.

8.7.3 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 11.11, "Memory Type Range Registers (MTRRs)," for information on setting up MTRRs.

8.7.4 Page Attribute Table (PAT)

Each logical processor has its own PAT MSR (IA32_PAT). However, as described in Section 11.12, "Page Attribute Table (PAT)," the PAT MSR settings must be the same for all processors in a system, including the logical processors.

8.7.5 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 15, "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.

8.7.6 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.

8.7.7 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 19.6.4, "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.

8.7.8 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.

8.7.9 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 8.2, "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.

8.7.10 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.

8.7.11 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 9.11.6.3.

8.7.12 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 8.1.3, "Handling Self- and Cross-Modifying Code," for a description of the requirements for self- and cross-modifying code in an IA-32 processor.

8.7.13 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.

8.7.13.1 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.

8.7.13.2 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 4.10, "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.

8.7.13.3 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 14.8, "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.

8.7.13.4 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.

8.8 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.

8.8.1 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 8.4, "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 8.9.5, "Identifying Topological Relationships in a MP System").

8.8.2 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 11.11, "Memory Type Range Registers (MTRRs)."

8.8.3 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 19.6.4, "Performance Monitoring and Intel Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture."

8.8.4 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*.

8.8.5 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 9.11.6.3.

8.9 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 8.7, "Intel® Hyper-Threading Technology Architecture," and Section 8.8, "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.

8.9.1 Hierarchical Mapping of Shared Resources

The APIC_ID value associated with each logical processor in a multi-processor system is unique (see Section 8.6, "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 level 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.
- **Tile** — A set of cores, possibly within modules, 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 level. The CORE_ID sub-field distinguishes processor cores. For a single-core processor, the width of this bit field is 0.
- **SMT** — A processor core provides one or more logical processors sharing execution resources. The SMT_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.

SMT and CORE sub-fields are bit-wise contiguous in the APIC_ID field (see Figure 8-5).

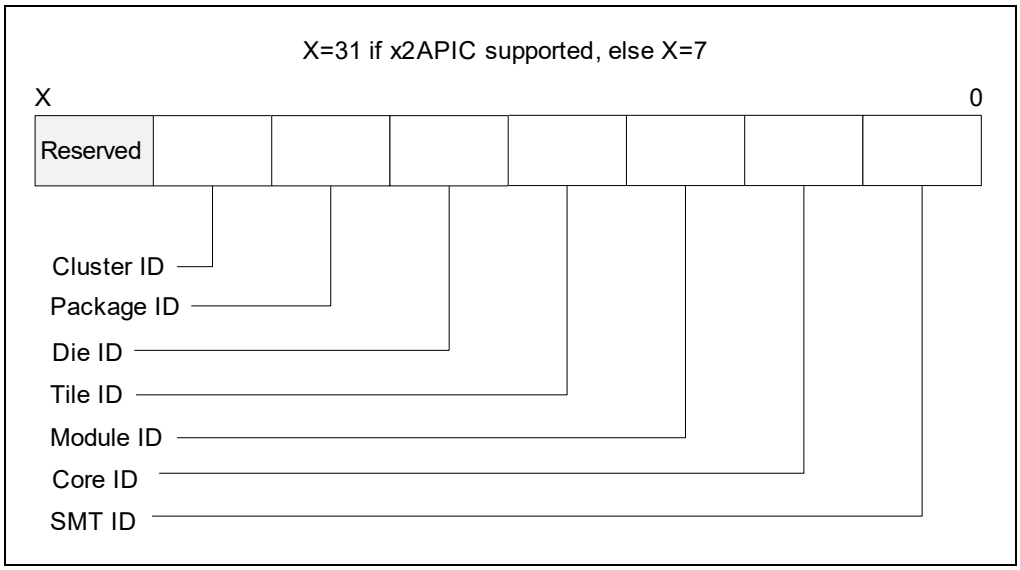


Figure 8-5. Generalized Seven Level Interpretation of the APIC ID

If the processor supports CPUID leaf 0BH and leaf 1FH, the 32-bit APIC ID can represent cluster plus several levels of topology within the physical processor package. The exact number of hierarchical levels within a physical processor package must be enumerated through CPUID leaf 0BH and leaf 1FH. Common processor families may employ topology similar to that represented by 8-bit Initial APIC ID. In general, CPUID leaf 0BH and leaf 1FH can support a topology enumeration algorithm that decompose a 32-bit APIC ID into more than four sub-fields (see Figure 8-6).

NOTE

CPUID leaf 0BH and leaf 1FH can have differences in the number of level types reported (CPUID leaf 1FH defines additional level types). If the processor supports CPUID leaf 1FH, usage of this leaf is preferred over leaf 0BH. CPUID leaf 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 8-16 through Example 8-21).

Figure 7-6 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.

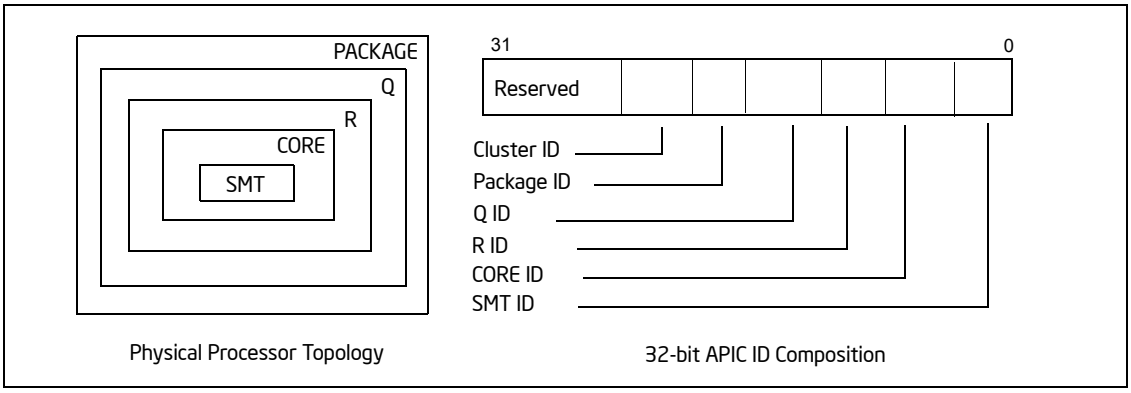


Figure 8-6. Conceptual Six-Level Topology and 32-bit APIC ID Composition

8.9.2 Hierarchical Mapping of CPUID Extended Topology Leaf

CPUID leaf 0BH and leaf 1FH provide enumeration parameters for software to identify each hierarchy of the processor topology in a deterministic manner. Each hierarchical level of the topology starting from the SMT level is represented numerically by a sub-leaf index within the CPUID 0BH leaf and 1FH leaf. Each level of the topology is mapped to a sub-field in the APIC ID, following the general relationship depicted in Figure 8-6. This mechanism allows software to query the exact number of levels 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 ECX until CPUID.(EAX=0BH or 1FH, ECX=N):ECX[15:8] returns an invalid "level type" encoding. The number of levels within the physical processor package is "N" (excluding PACKAGE). Using Figure 8-6 as an example, CPUID.(EAX=0BH or 1FH, ECX=4):ECX[15:8] will report 00H, indicating sub leaf 04H is invalid. This is also depicted by a pseudo code example:

Example 8-16. Number of Levels Below the Physical Processor Package

```
Byte type = 1;
    s = 0;
    While ( type ) {
        EAX = 0BH or 1FH; // query each sub leaf of CPUID leaf 0BH or 1FH; CPUID leaf 1FH is preferred over leaf 0BH if available
        ECX = s;
        CPUID;
        type = ECX[15:8]; // examine level type encoding
        s ++;
    }
N = ECX[7:0];
```

- Sub-leaf index 0 (ECX= 0 as input) provides enumeration parameters to extract the SMT sub-field of x2APIC ID. If EAX = 0BH or 1FH, and ECX =0 is specified as input when executing CPUID, CPUID.(EAX=0BH or 1FH, ECX=0):EAX[4:0] 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 SMT level. This value also corresponds to the bit-width of the sub-field of x2APIC ID corresponding the hierarchical level with sub-leaf index 0.
- For each subsequent higher sub-leaf index m, CPUID.(EAX=0BH or 1FH, ECX=m):EAX[4:0] reports the right-shift count that will allow software to extract part of x2APIC ID to distinguish higher-level topological entities. This means the right-shift value at of sub-leaf m, corresponds to the least significant (m+1) subfields of the 32-bit x2APIC ID.

Example 8-17. BitWidth Determination of x2APIC ID Subfields

```
For m = 0, m < N, m ++;
{    cumulative_width[m] = CPUID.(EAX=0BH or 1FH, ECX= 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 leaf 1FH is a preferred superset to leaf 0BH. Leaf 1FH defines additional level types, and it must be parsed by an algorithm that can handle the addition of future level types.

Previously, only the following encoding of hierarchical level types were defined: 0 (invalid), 1 (SMT), and 2 (core). With the additional hierarchical level types available (see Section 8.9.1, "Hierarchical Mapping of Shared Resources" and Figure 8-5, "Generalized Seven Level Interpretation of the APIC ID") software must not assume any "level type" encoding value to be related to any sub-leaf index, except sub-leaf 0.

Example 8-18. 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 level types (level types may or may not be known/supported by the software)
//
// Below is the list of sample level types used in the example.
// Refer to the CPUID Leaf 1FH definition for the actual level type numbers: "V2 Extended Topology Enumeration Leaf" .
//
// SMT
// CORE
// MODULE
// TILE
// DIE
// PACKAGE
//
// The example shows how to identify and derive the extraction bitmask for the levels with identify type SMT/CORE/DIE/PACKAGE
//

int DeriveSMT_Mask_Offsets (void)
{
    IF (!HWMTSupported()) return -1;
    execute cpuid with EAX = 0BH or 1FH, ECX = 0;
    IF (returned level type encoding in EXC[15:8] does not match SMT) return -1;
    Mask_SMT_shift = EAX[4:0];                    //# bits shift right of APIC ID to distinguish different cores
    SMT_MASK =~( (-1) << Mask_SMT_shift);         //shift left to derive extraction bitmask for SMT_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 = 0BH or 1FH, ECX = 0;
    WHILE( ECX[15:8] ) {                          //level type encoding is valid
        Mask_last_known_shift = EAX[4:0]
        IF (returned level type encoding in ECX[15:8] matches CORE) {
            Mask_Core_shift = EAX[4:0];
        }
        ELSE IF (returned level type encoding in ECX[15:8] matches DIE {
            Mask_Die_shift = EAX[4:0];
        }
        //
        // Keep enumerating. Check if the next level is the desired level and if not, keep enumerating until you reach a known level
        // or the invalid level ("0" level type). If there are more levels between DIE and PACKAGE, the unknown levels will be ignored
        // and treated as an extension of the last known level (i.e., DIE in this case).
        //

        ECX++;
        execute cpuid with EAX = 0BH or 1FH;
    }

    COREPlusSMT_MASK = ~( (-1) << Mask_Core_shift);
    DIEPlusCORE_MASK = ~( (-1) << Mask_Die_shift);

    //
    // Treat levels between DIE and physical package as an extension of DIE for software choosing not to implement or recognize
    // these unknown levels.
    //

    CORE_MASK = COREPlusSMT_MASK ^ SMT_MASK;
    DIE_MASK = DIEPlusCORE_MASK ^ COREPlusSMT_MASK;
    PACKAGE_MASK = (-1) << Mask_last_known_shift;

    return -1;

}
```

8.9.3 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 leaf 0BH) that is unique for each logical processor following power-up or RESET (see Section 8.6.1). 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 8-23) is limited to the number of logical processors enabled at runtime by the OS boot process.

Table 8-2 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.

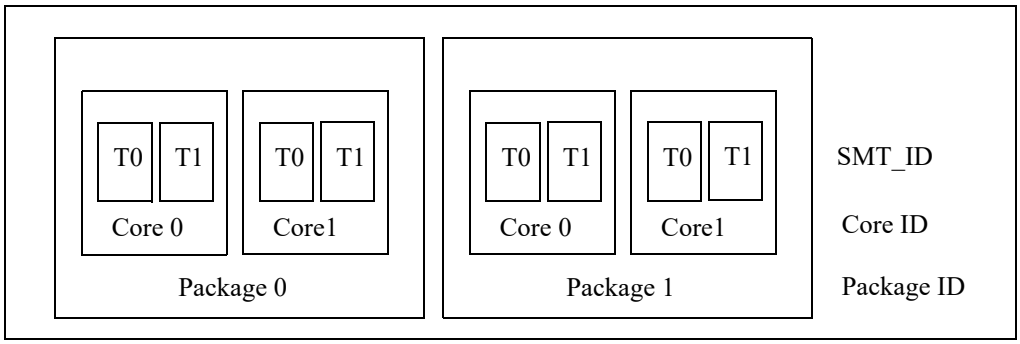


Figure 8-7. Topological Relationships between Hierarchical IDs in a Hypothetical MP Platform

Table 8-2. Initial APIC IDs for the Logical Processors in a System that has Four Intel Xeon MP Processors Supporting Intel Hyper-Threading Technology[1]

Initial APIC ID	Package ID	Core ID	SMT 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 8-3 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 8-3. 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	SMT 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

8.9.3.1 Hierarchical ID of Logical Processors with x2APIC ID

Table 8-4 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 8-4. 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	SMT 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

8.9.4 Algorithm for Three-Level Mappings of APIC_ID

Software can gather the initial APIC_IDs for each logical processor supported by the operating system at runtime[1] and extract identifiers corresponding to the three levels of sharing topology (package, core, and SMT). The three-level 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 leaf 0BH, CPUID.1:EBX[23:16]), CPUID.4[2]:EAX[31:26], and CPUID.4[3]: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 level hierarchy if it was developed to understand only three levels. However, software implementation needs to ensure it does not break if it runs on systems that have more levels in the hierarchy even if it does not recognize them.

1. As noted in Section 8.6 and Section 8.9.3, 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 start 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.

The extraction algorithm (for three-level mappings from an APIC ID) uses the general procedure depicted in Example 8-19, 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 level of the topology. The algorithm to derive extraction bit masks for SMT_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 leaf 0BH, each APIC ID contains a 32-bit value, the topology enumeration parameters needed to derive three-level extraction bit masks are:
    a. Query the right-shift value for the SMT level of the topology using CPUID leaf 0BH with ECX =0H as input. The number of bits to shift-right on x2APIC ID (EAX[4:0]) can distinguish different higher-level entities above SMT (e.g. processor cores) in the same physical package. This is also the width of the bit mask to extract the SMT_ID.
    b. Enumerate until the desired level is found (i.e. processor cores). Determine if the next level is the expected level. If the next level is not known to the software, keep enumerating until the next known or the last level. Software should use the previous level before this to represent the last previously known level (i.e. processor cores). If the software does not recognize or implement certain hierarchical levels, it should assume these unknown levels as an extension of the last known level.
    c. Query CPUID leaf 0BH for the amount of bit shift to distinguish next higher-level entities (e.g. physical processor packages) in the system. This describes an explicit three-level-topology situation for commonly available processors. Consult Example 8-17 to adapt to situations beyond three-level 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 SMT 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 SMT_ID, CORE_ID, and PACKAGE_ID, starting from SMT_ID.
    f. Apply each extraction bit mask to the 32-bit x2APIC ID to extract sub-field IDs.
4. If the processor does not support CPUID leaf 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.1: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 SMT_ID, CORE_ID, and PACKAGE_ID, starting from SMT_ID.
    e. Apply each extraction bit mask to the 8-bit initial APIC ID to extract sub-field IDs.

Example 8-19. 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 8-20. 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 DeriveSMT_Mask_Offsets (void)
{
     if (!HWMTSupported()) return -1;
     execute cpuid with eax = 11, ECX = 0;
     If (returned level type encoding in ECX[15:8] does not match SMT) return -1;
     Mask_SMT_shift = EAX[4:0]; // # bits shift right of APIC ID to distinguish different cores
     SMT_MASK = ~( (-1) << Mask_SMT_shift); // shift left to derive extraction bitmask for SMT_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] ) {            // level type encoding is valid
        Mask_Core_shift = EAX[4:0];      // needed to distinguish different physical packages
        ECX ++;
        execute cpuid with eax = 11;
   }
    COREPlusSMT_MASK = ~( (-1) << Mask_Core_shift);
    // treat levels between core and physical package as a core for software choosing not to implement or recognize
    // these unknown levels
    CORE_MASK = COREPlusSMT_MASK ^ SMT_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 8-21. 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.1.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.1.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 SMT_ID can be calculated from MaxLPIDsPerPackage()/MaxCoreIDsPerPackage()
// Then use the routine below to derive the corresponding width of SMT 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 SMT_ID, CORE_ID, and PACKAGE_ID respectively from the init APIC_ID
// To extract SMT_ID, MaxSubIDvalue is set to the address size of SMT_ID, Shift_Count = 0
// To extract CORE_ID, MaxSubIDvalue is the address size of CORE_ID, Shift_Count is width of SMT 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 level can be extracted from an 8-bit APIC_ID using the support routines illustrated in Example 8-21. The appropriate bit mask and shift value to construct the appropriate bit mask for each level must be determined dynamically at runtime.

8.9.5 Identifying Topological Relationships in a 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-level 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 8-22 and support routines shown in Example 8-19):

- The extraction start from the right-most bit field, corresponding to SMT_ID, the innermost hierarchy in a three-level topology (See Figure 8-7). 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-level identifiers of SMT_ID, CORE_ID, PACKAGE_IDs into arrays for each enabled logical processor. This is shown in Example 8-23a.
- 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 8-23b. 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 8-23. This example also depicts a technique to construct a mask to represent the logical processors that reside in the same core.

In Example 8-22, 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 8-22. Pseudo Code Depicting Three-level Extraction Algorithm

```
For Each local_APIC_ID{
    // Calculate SMT_MASK, the bit mask pattern to extract SMT_ID,
    // SMT_MASK is determined using topology enumertaion parameters
    // from CPUID leaf 0BH (Example 8-20);
    // otherwise, SMT_MASK is determined using CPUID leaf 01H and leaf 04H (Example 8-21).
    // This algorithm assumes there is symmetry across core boundary, i.e. each core within a
    //  package has the same number of logical processors
    // SMT_ID always starts from bit 0, corresponding to the right-most bit-field
    SMT_ID = APIC_ID & SMT_MASK;

// Extract CORE_ID:
    // CORE_MASK is determined in Example 8-20 or Example 8-21
    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 8-20 or Example 8-21
    PACKAGE_ID = (APIC_ID & PACKAGE_MASK) ;
}
```

Example 8-23. Compute the Number of Packages, Cores, and Processor Relationships in a MP System

a) Assemble lists of PACKAGE_ID, CORE_ID, and SMT_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-levels 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 SMT_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 8-20 or 8-bit ID in Example 8-21
                Extract the Package_ID, Core_ID and SMT_ID as explained in three level extraction
                      algorithm of Example 8-22
                PackageID[ProcessorNUM] = PACKAGE_ID;
                CoreID[ProcessorNum] = CORE_ID;
                SmtID[ProcessorNum] = SMT_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*).

8.10 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.

8.10.1 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 8.10.6.3, "Halt Idle Logical Processors," for more information about using the HLT instruction with processors supporting Intel Hyper-Threading Technology.)

8.10.2 PAUSE Instruction

The PAUSE instruction can improves 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 de-pipelines the spin-wait loop to prevent it from consuming execution resources excessively and consume power needlessly. (See Section 8.10.6.1, "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.)

8.10.3 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.1.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 8-24.

Example 8-24. 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;
}
```

8.10.4 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.

8.10.5 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 the CPUID monitor leaf function (EAX = 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*.

8.10.6 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*.

8.10.6.1 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 8.10.2, "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.

8.10.6.2 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 8-25:

Example 8-25. 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 >= IdleTimeThreshhold) 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 8-26. 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 >= IdleTimeThreshhold) 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
}
```

8.10.6.3 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.

8.10.6.4 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 8-27:

Example 8-27. 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 >= IdleTimeThreshhold) 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
        MWAIT          // EAX, ECX = 0
        }
}
```

8.10.6.5 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.

---

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.

8.10.6.6 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*.

8.10.6.7 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.

8.11 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 8.4, "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 busses are tied together. Larger systems are not supported.

8.11.1 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 8.4.1, "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 8-5 describes the various fields of the boot phase IPIs.

Table 8-5. 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).

8.11.2 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 8.4.5, "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 8-1).
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 8-1, 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.

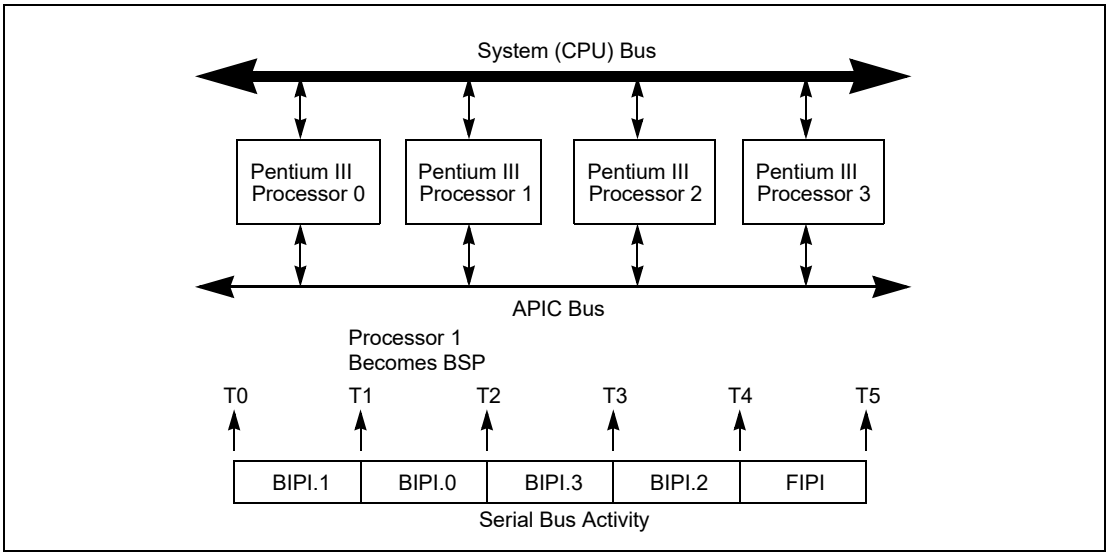


Figure 8-1. MP System With Multiple Pentium III Processors

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 8.4.4, “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.

8.11.2.1 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.

14. Updates to Chapter 10, Volume 3A

Change bars and green text show changes to Chapter 10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

Changes to this chapter: Update to Figure 10-4, "Local APIC Structure". Update to Section 10.4.3, "Enabling or Disabling the Local APIC", Section 10.5, "Handling Local Interrupts", and Section 10.5.1, "Local Vector Table".

CHAPTER 10
ADVANCED PROGRAMMABLE INTERRUPT CONTROLLER (APIC)

The Advanced Programmable Interrupt Controller (APIC), referred to in the following sections as the local APIC, was introduced into the IA-32 processors with the Pentium processor (see Section 22.27, "Advanced Programmable Interrupt Controller (APIC)") and is included in the P6 family, Pentium 4, Intel Xeon processors, and other more recent Intel 64 and IA-32 processor families (see Section 10.4.2, "Presence of the Local APIC"). The local APIC performs two primary functions for the processor:

- It receives interrupts from the processor's interrupt pins, from internal sources and from an external I/O APIC (or other external interrupt controller). It sends these to the processor core for handling.
- In multiple processor (MP) systems, it sends and receives interprocessor interrupt (IPI) messages to and from other logical processors on the system bus. IPI messages can be used to distribute interrupts among the processors in the system or to execute system wide functions (such as, booting up processors or distributing work among a group of processors).

The external **I/O APIC** is part of Intel's system chip set. Its primary function is to receive external interrupt events from the system and its associated I/O devices and relay them to the local APIC as interrupt messages. In MP systems, the I/O APIC also provides a mechanism for distributing external interrupts to the local APICs of selected processors or groups of processors on the system bus.

This chapter provides a description of the local APIC and its programming interface. It also provides an overview of the interface between the local APIC and the I/O APIC. Contact Intel for detailed information about the I/O APIC.

When a local APIC has sent an interrupt to its processor core for handling, the processor uses the interrupt and exception handling mechanism described in Chapter 6, "Interrupt and Exception Handling." See Section 6.1, "Interrupt and Exception Overview," for an introduction to interrupt and exception handling.

10.1 LOCAL AND I/O APIC OVERVIEW

Each local APIC consists of a set of APIC registers (see Table 10-1) and associated hardware that control the delivery of interrupts to the processor core and the generation of IPI messages. The APIC registers are memory mapped and can be read and written to using the MOV instruction.

Local APICs can receive interrupts from the following sources:

- **Locally connected I/O devices** — These interrupts originate as an edge or level asserted by an I/O device that is connected directly to the processor's local interrupt pins (LINT0 and LINT1). The I/O devices may also be connected to an 8259-type interrupt controller that is in turn connected to the processor through one of the local interrupt pins.
- **Externally connected I/O devices** — These interrupts originate as an edge or level asserted by an I/O device that is connected to the interrupt input pins of an I/O APIC. Interrupts are sent as I/O interrupt messages from the I/O APIC to one or more of the processors in the system.
- **Inter-processor interrupts (IPIs)** — An Intel 64 or IA-32 processor can use the IPI mechanism to interrupt another processor or group of processors on the system bus. IPIs are used for software self-interrupts, interrupt forwarding, or preemptive scheduling.
- **APIC timer generated interrupts** — The local APIC timer can be programmed to send a local interrupt to its associated processor when a programmed count is reached (see Section 10.5.4, "APIC Timer").
- **Performance monitoring counter interrupts** — P6 family, Pentium 4, and Intel Xeon processors provide the ability to send an interrupt to its associated processor when a performance-monitoring counter overflows (see Section 19.6.3.5.8, "Generating an Interrupt on Overflow").
- **Thermal Sensor interrupts** — Pentium 4 and Intel Xeon processors provide the ability to send an interrupt to themselves when the internal thermal sensor has been tripped (see Section 14.8.2, "Thermal Monitor").

- **APIC internal error interrupts** — When an error condition is recognized within the local APIC (such as an attempt to access an unimplemented register), the APIC can be programmed to send an interrupt to its associated processor (see Section 10.5.3, "Error Handling").

Of these interrupt sources: the processor's LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, the thermal sensor, and the internal APIC error detector are referred to as **local interrupt sources**. Upon receiving a signal from a local interrupt source, the local APIC delivers the interrupt to the processor core using an interrupt delivery protocol that has been set up through a group of APIC registers called the **local vector table** or **LVT** (see Section 10.5.1, "Local Vector Table"). A separate entry is provided in the local vector table for each local interrupt source, which allows a specific interrupt delivery protocol to be set up for each source. For example, if the LINT1 pin is going to be used as an NMI pin, the LINT1 entry in the local vector table can be set up to deliver an interrupt with vector number 2 (NMI interrupt) to the processor core.

The local APIC handles interrupts from the other two interrupt sources (externally connected I/O devices and IPIs) through its IPI message handling facilities.

A processor can generate IPIs by programming the interrupt command register (ICR) in its local APIC (see Section 10.6.1, "Interrupt Command Register (ICR)"). The act of writing to the ICR causes an IPI message to be generated and issued on the system bus (for Pentium 4 and Intel Xeon processors) or on the APIC bus (for Pentium and P6 family processors). See Section 10.2, "System Bus Vs. APIC Bus."

IPIs can be sent to other processors in the system or to the originating processor (self-interrupts). When the target processor receives an IPI message, its local APIC handles the message automatically (using information included in the message such as vector number and trigger mode). See Section 10.6, "Issuing Interprocessor Interrupts," for a detailed explanation of the local APIC's IPI message delivery and acceptance mechanism.

The local APIC can also receive interrupts from externally connected devices through the I/O APIC (see Figure 10-1). The I/O APIC is responsible for receiving interrupts generated by system hardware and I/O devices and forwarding them to the local APIC as interrupt messages.

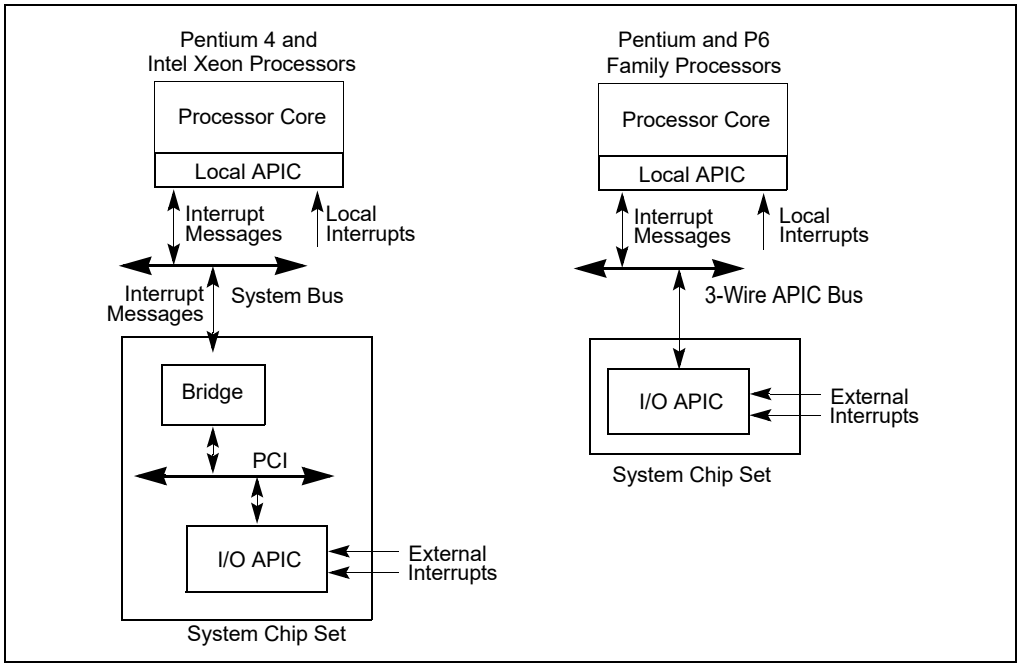


Figure 10-1. Relationship of Local APIC and I/O APIC In Single-Processor Systems

Individual pins on the I/O APIC can be programmed to generate a specific interrupt vector when asserted. The I/O APIC also has a "virtual wire mode" that allows it to communicate with a standard 8259A-style external interrupt controller. Note that the local APIC can be disabled (see Section 10.4.3, "Enabling or Disabling the Local APIC"). This allows an associated processor core to receive interrupts directly from an 8259A interrupt controller.

Both the local APIC and the I/O APIC are designed to operate in MP systems (see Figures 10-2 and 10-3). Each local APIC handles interrupts from the I/O APIC, IPIs from processors on the system bus, and self-generated interrupts. Interrupts can also be delivered to the individual processors through the local interrupt pins; however, this mechanism is commonly not used in MP systems.

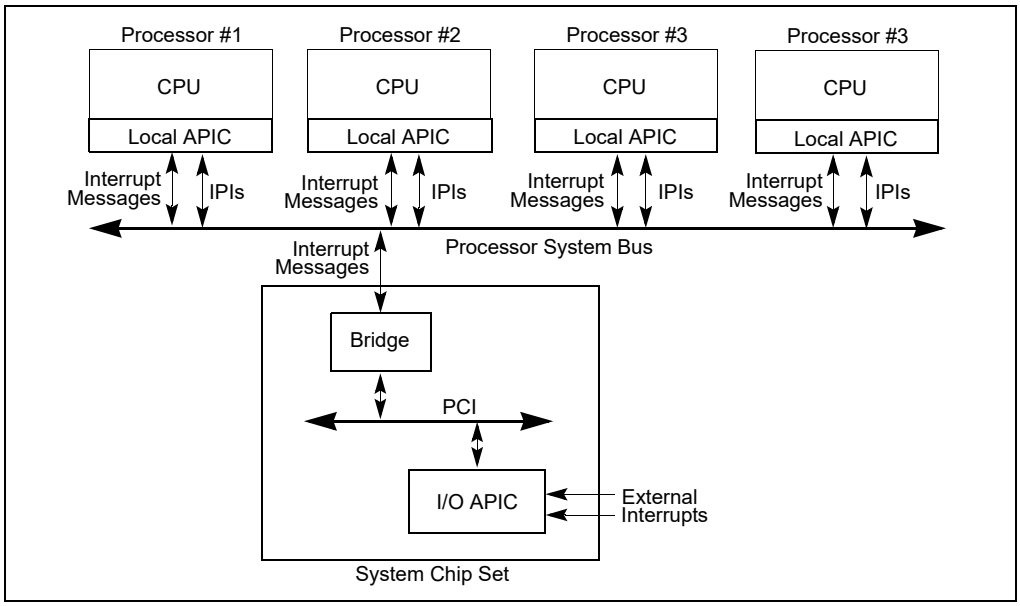


Figure 10-2. Local APICs and I/O APIC When Intel Xeon Processors Are Used in Multiple-Processor Systems

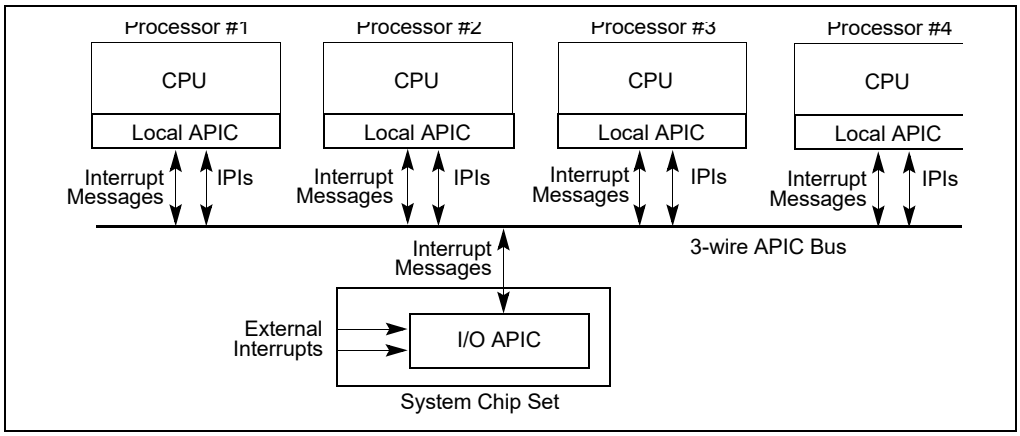


Figure 10-3. Local APICs and I/O APIC When P6 Family Processors Are Used in Multiple-Processor Systems

The IPI mechanism is typically used in MP systems to send fixed interrupts (interrupts for a specific vector number) and special-purpose interrupts to processors on the system bus. For example, a local APIC can use an IPI to forward a fixed interrupt to another processor for servicing. Special-purpose IPIs (including NMI, INIT, SMI and SIPI IPIs) allow one or more processors on the system bus to perform system-wide boot-up and control functions.

The following sections focus on the local APIC and its implementation in the Pentium 4, Intel Xeon, and P6 family processors. In these sections, the terms "local APIC" and "I/O APIC" refer to local and I/O APICs used with the P6 family processors and to local and I/O xAPICs used with the Pentium 4 and Intel Xeon processors (see Section 10.3, "The Intel® 82489DX External APIC, the APIC, the xAPIC, and the X2APIC").

10.2 SYSTEM BUS VS. APIC BUS

For the P6 family and Pentium processors, the I/O APIC and local APICs communicate through the 3-wire inter-APIC bus (see Figure 10-3). Local APICs also use the APIC bus to send and receive IPIs. The APIC bus and its messages are invisible to software and are not classed as architectural.

Beginning with the Pentium 4 and Intel Xeon processors, the I/O APIC and local APICs (using the xAPIC architecture) communicate through the system bus (see Figure 10-2). The I/O APIC sends interrupt requests to the processors on the system bus through bridge hardware that is part of the Intel chip set. The bridge hardware generates the interrupt messages that go to the local APICs. IPIs between local APICs are transmitted directly on the system bus.

10.3 THE INTEL® 82489DX EXTERNAL APIC, THE APIC, THE XAPIC, AND THE X2APIC

The local APIC in the P6 family and Pentium processors is an architectural subset of the Intel® 82489DX external APIC. See Section 22.27.1, "Software Visible Differences Between the Local APIC and the 82489DX."

The APIC architecture used in the Pentium 4 and Intel Xeon processors (called the xAPIC architecture) is an extension of the APIC architecture found in the P6 family processors. The primary difference between the APIC and xAPIC architectures is that with the xAPIC architecture, the local APICs and the I/O APIC communicate through the system bus. With the APIC architecture, they communication through the APIC bus (see Section 10.2, "System Bus Vs. APIC Bus"). Also, some APIC architectural features have been extended and/or modified in the xAPIC architecture. These extensions and modifications are described in Section 10.4 through Section 10.10.

The basic operating mode of the xAPIC is **xAPIC mode**. The x2APIC architecture is an extension of the xAPIC architecture, primarily to increase processor addressability. The x2APIC architecture provides backward compatibility to the xAPIC architecture and forward extendability for future Intel platform innovations. These extensions and modifications are supported by a new mode of execution (**x2APIC mode**) are detailed in Section 10.12.

10.4 LOCAL APIC

The following sections describe the architecture of the local APIC and how to detect it, identify it, and determine its status. Descriptions of how to program the local APIC are given in Section 10.5.1, "Local Vector Table," and Section 10.6.1, "Interrupt Command Register (ICR)."

10.4.1 The Local APIC Block Diagram

Figure 10-4 gives a functional block diagram for the local APIC. Software interacts with the local APIC by reading and writing its registers. APIC registers are memory-mapped to a 4-KByte region of the processor's physical address space with an initial starting address of FEE00000H. For correct APIC operation, this address space must be mapped to an area of memory that has been designated as strong uncacheable (UC). See Section 11.3, "Methods of Caching Available."

In MP system configurations, the APIC registers for Intel 64 or IA-32 processors on the system bus are initially mapped to the same 4-KByte region of the physical address space. Software has the option of changing initial mapping to a different 4-KByte region for all the local APICs or of mapping the APIC registers for each local APIC to its own 4-KByte region. Section 10.4.5, "Relocating the Local APIC Registers," describes how to relocate the base address for APIC registers.

On processors supporting x2APIC architecture (indicated by CPUID.01H:ECX[21] = 1), the local APIC supports operation both in xAPIC mode and (if enabled by software) in x2APIC mode. x2APIC mode provides extended processor addressability (see Section 10.12).

NOTE

For P6 family, Pentium 4, and Intel Xeon processors, the APIC handles all memory accesses to addresses within the 4-KByte APIC register space internally and no external bus cycles are produced. For the Pentium processors with an on-chip APIC, bus cycles are produced for accesses to the APIC register space. Thus, for software intended to run on Pentium processors, system software should explicitly not map the APIC register space to regular system memory. Doing so can result in an invalid opcode exception (#UD) being generated or unpredictable execution.

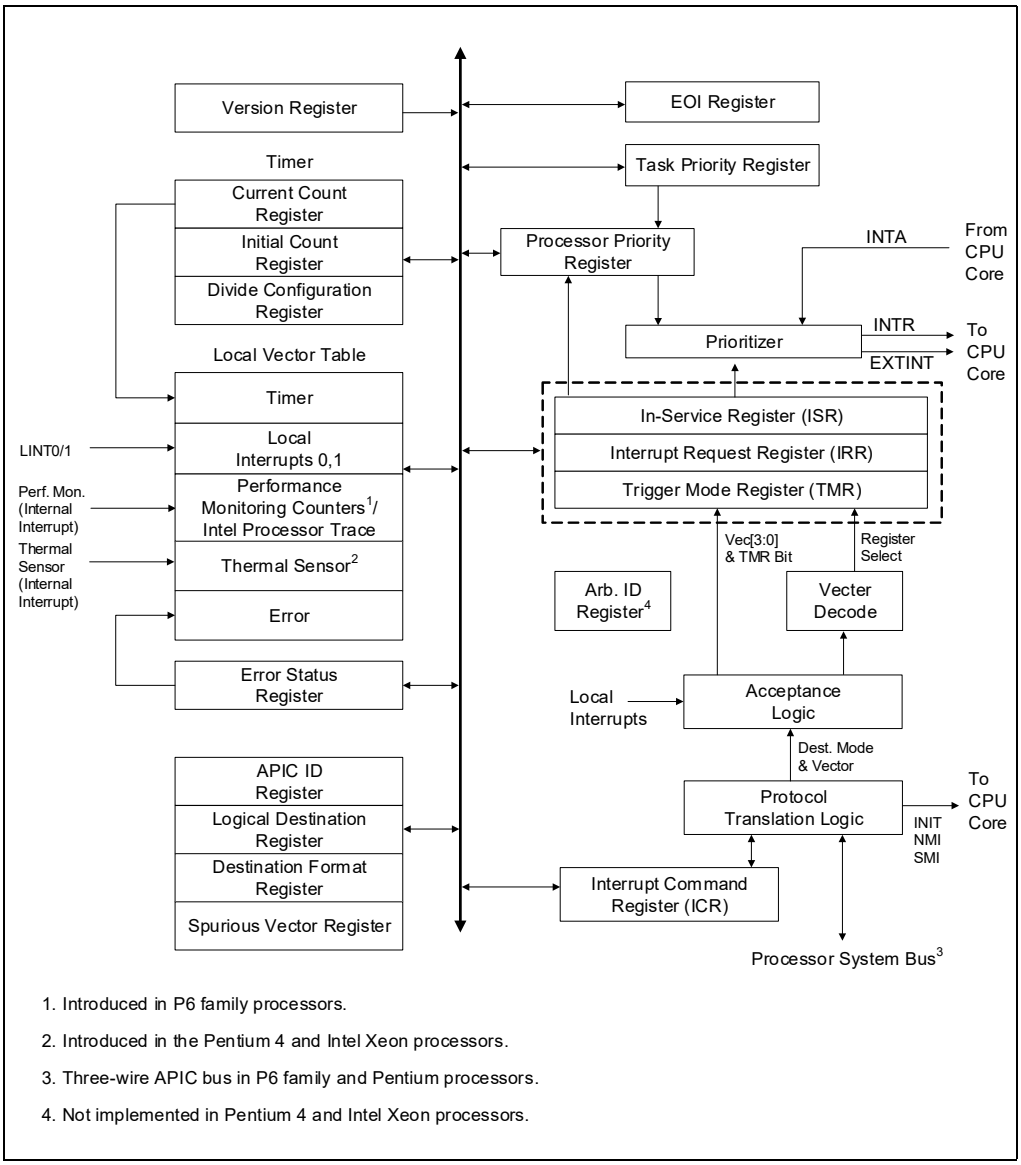


Figure 10-4. Local APIC Structure

Table 10-1 shows how the APIC registers are mapped into the 4-KByte APIC register space. Registers are 32 bits, 64 bits, or 256 bits in width; all are aligned on 128-bit boundaries. All 32-bit registers should be accessed using 128-bit aligned 32-bit loads or stores. Some processors may support loads and stores of less than 32 bits to some of the APIC registers. This is model specific behavior and is not guaranteed to work on all processors. Any FP/MMX/SSE access to an APIC register, or any access that touches bytes 4 through 15 of an APIC register may cause undefined behavior and must not be executed. This undefined behavior could include hangs, incorrect results or unexpected exceptions, including machine checks, and may vary between implementations. Wider registers (64-bit or 256-bit) must be accessed using multiple 32-bit loads or stores, with all accesses being 128-bit aligned.

The local APIC registers listed in Table 10-1 are not MSRs. The only MSR associated with the programming of the local APIC is the IA32_APIC_BASE MSR (see Section 10.4.3, "Enabling or Disabling the Local APIC").

NOTE

In processors based on Nehalem[1] microarchitecture, the Local APIC ID Register is no longer Read/Write; it is Read Only.

Table 10-1. Local APIC Register Address Map

Address	Register Name	Software Read/Write
FEE0 0000H	Reserved	
FEE0 0010H	Reserved	
FEE0 0020H	Local APIC ID Register	Read/Write.
FEE0 0030H	Local APIC Version Register	Read Only.
FEE0 0040H	Reserved	
FEE0 0050H	Reserved	
FEE0 0060H	Reserved	
FEE0 0070H	Reserved	
FEE0 0080H	Task Priority Register (TPR)	Read/Write.
FEE0 0090H	Arbitration Priority Register[1] (APR)	Read Only.
FEE0 00A0H	Processor Priority Register (PPR)	Read Only.
FEE0 00B0H	EOI Register	Write Only.
FEE0 00C0H	Remote Read Register[1] (RRD)	Read Only
FEE0 00D0H	Logical Destination Register	Read/Write.
FEE0 00E0H	Destination Format Register	Read/Write (see Section 10.6.2.2).
FEE0 00F0H	Spurious Interrupt Vector Register	Read/Write (see Section 10.9.
FEE0 0100H	In-Service Register (ISR); bits 31:0	Read Only.
FEE0 0110H	In-Service Register (ISR); bits 63:32	Read Only.
FEE0 0120H	In-Service Register (ISR); bits 95:64	Read Only.
FEE0 0130H	In-Service Register (ISR); bits 127:96	Read Only.
FEE0 0140H	In-Service Register (ISR); bits 159:128	Read Only.
FEE0 0150H	In-Service Register (ISR); bits 191:160	Read Only.
FEE0 0160H	In-Service Register (ISR); bits 223:192	Read Only.
FEE0 0170H	In-Service Register (ISR); bits 255:224	Read Only.
FEE0 0180H	Trigger Mode Register (TMR); bits 31:0	Read Only.
FEE0 0190H	Trigger Mode Register (TMR); bits 63:32	Read Only.
FEE0 01A0H	Trigger Mode Register (TMR); bits 95:64	Read Only.
FEE0 01B0H	Trigger Mode Register (TMR); bits 127:96	Read Only.
FEE0 01C0H	Trigger Mode Register (TMR); bits 159:128	Read Only.
FEE0 01D0H	Trigger Mode Register (TMR); bits 191:160	Read Only.

1. See Table 2-1, "CPUID Signature Values of DisplayFamily_DisplayModel," on page 1, and Section 2.8, "MSRs In the Nehalem Microarchitecture" of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4* to determine which processors are based on Nehalem microarchitecture.

Table 10-1. Local APIC Register Address Map (Contd.)

Address	Register Name	Software Read/Write
FEE0 01E0H	Trigger Mode Register (TMR); bits 223:192	Read Only.
FEE0 01F0H	Trigger Mode Register (TMR); bits 255:224	Read Only.
FEE0 0200H	Interrupt Request Register (IRR); bits 31:0	Read Only.
FEE0 0210H	Interrupt Request Register (IRR); bits 63:32	Read Only.
FEE0 0220H	Interrupt Request Register (IRR); bits 95:64	Read Only.
FEE0 0230H	Interrupt Request Register (IRR); bits 127:96	Read Only.
FEE0 0240H	Interrupt Request Register (IRR); bits 159:128	Read Only.
FEE0 0250H	Interrupt Request Register (IRR); bits 191:160	Read Only.
FEE0 0260H	Interrupt Request Register (IRR); bits 223:192	Read Only.
FEE0 0270H	Interrupt Request Register (IRR); bits 255:224	Read Only.
FEE0 0280H	Error Status Register	Read Only.
FEE0 0290H through FEE0 02E0H	Reserved	
FEE0 02F0H	LVT Corrected Machine Check Interrupt (CMCI) Register	Read/Write.
FEE0 0300H	Interrupt Command Register (ICR); bits 0-31	Read/Write.
FEE0 0310H	Interrupt Command Register (ICR); bits 32-63	Read/Write.
FEE0 0320H	LVT Timer Register	Read/Write.
FEE0 0330H	LVT Thermal Sensor Register[2]	Read/Write.
FEE0 0340H	LVT Performance Monitoring Counters Register[3]	Read/Write.
FEE0 0350H	LVT LINT0 Register	Read/Write.
FEE0 0360H	LVT LINT1 Register	Read/Write.
FEE0 0370H	LVT Error Register	Read/Write.
FEE0 0380H	Initial Count Register (for Timer)	Read/Write.
FEE0 0390H	Current Count Register (for Timer)	Read Only.
FEE0 03A0H through FEE0 03D0H	Reserved	
FEE0 03E0H	Divide Configuration Register (for Timer)	Read/Write.
FEE0 03F0H	Reserved	

NOTES:

1. Not supported in the Pentium 4 and Intel Xeon processors. The Illegal Register Access bit (7) of the ESR will not be set when writing to these registers.
2. Introduced in the Pentium 4 and Intel Xeon processors. This APIC register and its associated function are implementation dependent and may not be present in future IA-32 or Intel 64 processors.
3. Introduced in the Pentium Pro processor. This APIC register and its associated function are implementation dependent and may not be present in future IA-32 or Intel 64 processors.

10.4.2 Presence of the Local APIC

Beginning with the P6 family processors, the presence or absence of an on-chip local APIC can be detected using the CPUID instruction. When the CPUID instruction is executed with a source operand of 1 in the EAX register, bit 9 of the CPUID feature flags returned in the EDX register indicates the presence (set) or absence (clear) of a local APIC.

10.4.3 Enabling or Disabling the Local APIC

The local APIC can be enabled or disabled in either of two ways:

1. Using the APIC global enable/disable flag in the IA32_APIC_BASE MSR (MSR address 1BH; see Figure 10-5):
   - When IA32_APIC_BASE[11] is 0, the processor is functionally equivalent to an IA-32 processor without an on-chip APIC. The CPUID feature flag for the APIC (see Section 10.4.2, "Presence of the Local APIC") is also set to 0.
   - When IA32_APIC_BASE[11] is set to 0, processor APICs based on the 3-wire APIC bus cannot be generally re-enabled until a system hardware reset. The 3-wire bus loses track of arbitration that would be necessary for complete re-enabling. Certain APIC functionality can be enabled (for example: performance and thermal monitoring interrupt generation).
   - For processors that use Front Side Bus (FSB) delivery of interrupts, software may disable or enable the APIC by setting and resetting IA32_APIC_BASE[11]. A hardware reset is not required to re-start APIC functionality, if software guarantees no interrupt will be sent to the APIC as IA32_APIC_BASE[11] is cleared.
   - When IA32_APIC_BASE[11] is set to 0, prior initialization to the APIC may be lost and the APIC may return to the state described in Section 10.4.7.1, "Local APIC State After Power-Up or Reset."
2. Using the APIC software enable/disable flag in the spurious-interrupt vector register (see Figure 10-23):
   - If IA32_APIC_BASE[11] is 1, software can temporarily disable a local APIC at any time by clearing the APIC software enable/disable flag in the spurious-interrupt vector register (see Figure 10-23). The state of the local APIC when in this software-disabled state is described in Section 10.4.7.2, "Local APIC State After It Has Been Software Disabled."
   - When the local APIC is in the software-disabled state, it can be re-enabled at any time by setting the APIC software enable/disable flag to 1.

For the Pentium processor, the APICEN pin (which is shared with the PICD1 pin) is used during power-up or reset to disable the local APIC.

Note that each entry in the LVT has a mask bit that can be used to inhibit interrupts from being delivered to the processor from selected local interrupt sources (the LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, Intel® Processor Trace, the thermal sensor, and/or the internal APIC error detector).

10.4.4 Local APIC Status and Location

The status and location of the local APIC are contained in the IA32_APIC_BASE MSR (see Figure 10-5). MSR bit functions are described below:

- **BSP flag, bit 8** — Indicates if the processor is the bootstrap processor (BSP). See Section 8.4, "Multiple-Processor (MP) Initialization." Following a power-up or reset, this flag is set to 1 for the processor selected as the BSP and set to 0 for the remaining processors (APs).
- **APIC Global Enable flag, bit 11** — Enables or disables the local APIC (see Section 10.4.3, "Enabling or Disabling the Local APIC"). This flag is available in the Pentium 4, Intel Xeon, and P6 family processors. It is not guaranteed to be available or available at the same location in future Intel 64 or IA-32 processors.
- **APIC Base field, bits 12 through 35** — Specifies the base address of the APIC registers. This 24-bit value is extended by 12 bits at the low end to form the base address. This automatically aligns the address on a 4-KByte boundary. Following a power-up or reset, the field is set to FEE0 0000H.
- Bits 0 through 7, bits 9 and 10, and bits MAXPHYADDR[1] through 63 in the IA32_APIC_BASE MSR are reserved.

1. The MAXPHYADDR is 36 bits for processors that do not support CPUID leaf 80000008H, or indicated by CPUID.80000008H:EAX[bits 7:0] for processors that support CPUID leaf 80000008H.

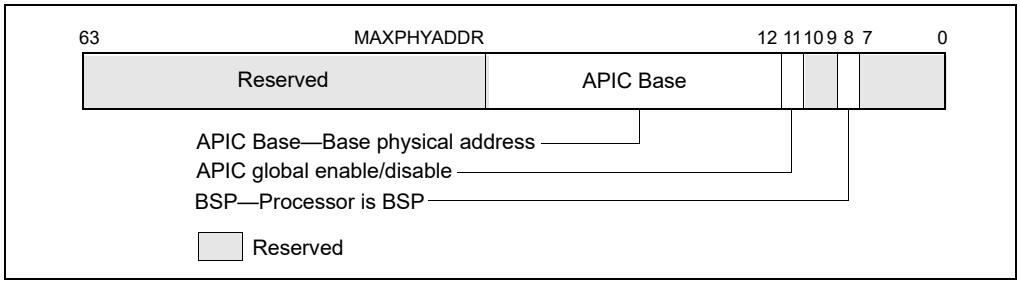


Figure 10-5. IA32_APIC_BASE MSR (APIC_BASE_MSR in P6 Family)

10.4.5 Relocating the Local APIC Registers

The Pentium 4, Intel Xeon, and P6 family processors permit the starting address of the APIC registers to be relocated from FEE00000H to another physical address by modifying the value in the base address field of the IA32_APIC_BASE MSR. This extension of the APIC architecture is provided to help resolve conflicts with memory maps of existing systems and to allow individual processors in an MP system to map their APIC registers to different locations in physical memory.

10.4.6 Local APIC ID

At power up, system hardware assigns a unique APIC ID to each local APIC on the system bus (for Pentium 4 and Intel Xeon processors) or on the APIC bus (for P6 family and Pentium processors). The hardware assigned APIC ID is based on system topology and includes encoding for socket position and cluster information (see Figure 8-2 and Section 8.9.1, "Hierarchical Mapping of Shared Resources").

In MP systems, the local APIC ID is also used as a processor ID by the BIOS and the operating system. Some processors permit software to modify the APIC ID. However, the ability of software to modify the APIC ID is processor model specific. Because of this, operating system software should avoid writing to the local APIC ID register. The value returned by bits 31-24 of the EBX register (when the CPUID instruction is executed with a source operand value of 1 in the EAX register) is always the Initial APIC ID (determined by the platform initialization). This is true even if software has changed the value in the Local APIC ID register.

The processor receives the hardware assigned APIC ID (or Initial APIC ID) by sampling pins A11# and A12# and pins BR0# through BR3# (for the Pentium 4, Intel Xeon, and P6 family processors) and pins BE0# through BE3# (for the Pentium processor). The APIC ID latched from these pins is stored in the APIC ID field of the local APIC ID register (see Figure 10-6), and is used as the Initial APIC ID for the processor.

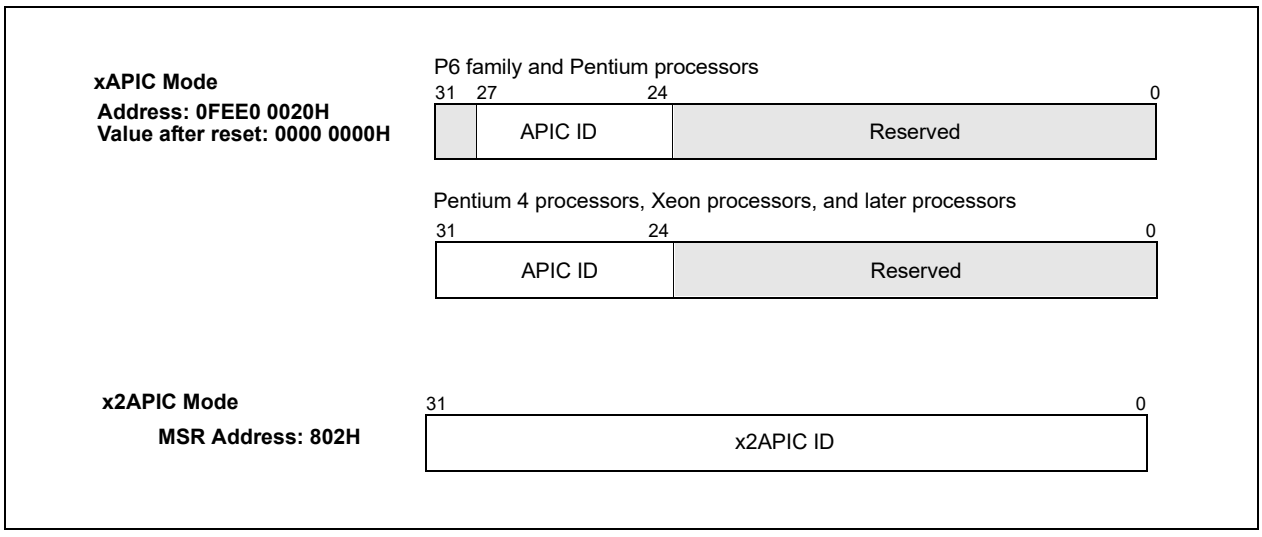


Figure 10-6. Local APIC ID Register

For the P6 family and Pentium processors, the local APIC ID field in the local APIC ID register is 4 bits. Encodings 0H through EH can be used to uniquely identify 15 different processors connected to the APIC bus. For the Pentium 4 and Intel Xeon processors, the xAPIC specification extends the local APIC ID field to 8 bits. These can be used to identify up to 255 processors in the system.

10.4.7 Local APIC State

The following sections describe the state of the local APIC and its registers following a power-up or reset, after the local APIC has been software disabled, following an INIT reset, and following an INIT-deassert message.

x2APIC will introduce 32-bit ID; see Section 10.12.

10.4.7.1 Local APIC State After Power-Up or Reset

Following a power-up or reset of the processor, the state of local APIC and its registers are as follows:

- The following registers are reset to all 0s.
    - IRR, ISR, TMR, ICR, LDR, and TPR.
    - Timer initial count and timer current count registers.
    - Divide configuration register.
- The DFR register is reset to all 1s.
- The LVT register is reset to 0s except for the mask bits; these are set to 1s.
- The local APIC version register is not affected.
- The local APIC ID register is set to a unique APIC ID. (Pentium and P6 family processors only). The Arb ID register is set to the value in the APIC ID register.
- The spurious-interrupt vector register is initialized to 000000FFH. By setting bit 8 to 0, software disables the local APIC.
- If the processor is the only processor in the system or it is the BSP in an MP system (see Section 8.4.1, "BSP and AP Processors"); the local APIC will respond normally to INIT and NMI messages, to INIT# signals and to STPCLK# signals. If the processor is in an MP system and has been designated as an AP; the local APIC will respond the same as for the BSP. In addition, it will respond to SIPI messages. For P6 family processors only, an AP will not respond to a STPCLK# signal.

10.4.7.2 Local APIC State After It Has Been Software Disabled

When the APIC software enable/disable flag in the spurious interrupt vector register has been explicitly cleared (as opposed to being cleared during a power up or reset), the local APIC is temporarily disabled (see Section 10.4.3, "Enabling or Disabling the Local APIC"). The operation and response of a local APIC while in this software-disabled state is as follows:

- The local APIC will respond normally to INIT, NMI, SMI, and SIPI messages.
- Pending interrupts in the IRR and ISR registers are held and require masking or handling by the CPU.
- The local APIC can still issue IPIs. It is software's responsibility to avoid issuing IPIs through the IPI mechanism and the ICR register if sending interrupts through this mechanism is not desired.
- The reception of any interrupt or transmission of any IPIs that are in progress when the local APIC is disabled are completed before the local APIC enters the software-disabled state.
- The mask bits for all the LVT entries are set. Attempts to reset these bits will be ignored.
- (For Pentium and P6 family processors) The local APIC continues to listen to all bus messages in order to keep its arbitration ID synchronized with the rest of the system.

10.4.7.3 Local APIC State After an INIT Reset ("Wait-for-SIPI" State)

An INIT reset of the processor can be initiated in either of two ways:

- By asserting the processor's INIT# pin.
- By sending the processor an INIT IPI (an IPI with the delivery mode set to INIT).

Upon receiving an INIT through either of these mechanisms, the processor responds by beginning the initialization process of the processor core and the local APIC. The state of the local APIC following an INIT reset is the same as it is after a power-up or hardware reset, except that the APIC ID and arbitration ID registers are not affected. This state is also referred to at the "wait-for-SIPI" state (see also: Section 8.4.2, "MP Initialization Protocol Requirements and Restrictions").

10.4.7.4 Local APIC State After It Receives an INIT-Deassert IPI

Only the Pentium and P6 family processors support the INIT-deassert IPI. An INIT-deassert IPI has no affect on the state of the APIC, other than to reload the arbitration ID register with the value in the APIC ID register.

10.4.8 Local APIC Version Register

The local APIC contains a hardwired version register. Software can use this register to identify the APIC version (see Figure 10-7). In addition, the register specifies the number of entries in the local vector table (LVT) for a specific implementation.

The fields in the local APIC version register are as follows:

Version The version numbers of the local APIC:

- 0XH — 82489DX discrete APIC.
- 10H - 15H — Integrated APIC.
- Other values reserved.

Max LVT Entry Shows the number of LVT entries minus 1. For the Pentium 4 and Intel Xeon processors (which have 6 LVT entries), the value returned in the Max LVT field is 5; for the P6 family processors (which have 5 LVT entries), the value returned is 4; for the Pentium processor (which has 4 LVT entries), the value returned is 3. For processors based on the Nehalem microarchitecture (which has 7 LVT entries) and onward, the value returned is 6.

Suppress EOI-broadcasts Indicates whether software can inhibit the broadcast of EOI message by setting bit 12 of the Spurious Interrupt Vector Register; see Section 10.8.5 and Section 10.9.

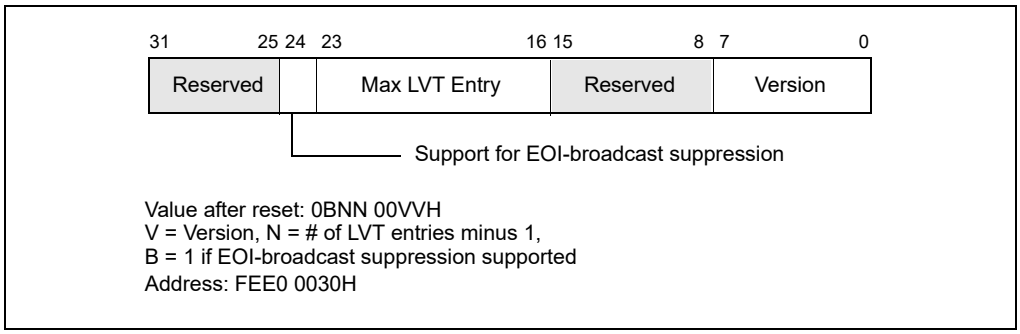


Figure 10-7. Local APIC Version Register

10.5 HANDLING LOCAL INTERRUPTS

The following sections describe facilities that are provided in the local APIC for handling local interrupts. These include: the processor's LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, Intel Processor Trace, the thermal sensor, and the internal APIC error detector. Local interrupt handling facilities include: the LVT, the error status register (ESR), the divide configuration register (DCR), and the initial count and current count registers.

10.5.1 Local Vector Table

The local vector table (LVT) allows software to specify the manner in which the local interrupts are delivered to the processor core. It consists of the following 32-bit APIC registers (see Figure 10-8), one for each local interrupt:

- **LVT CMCI Register (FEE0 02F0H)** — Specifies interrupt delivery when an overflow condition of corrected machine check error count reaching a threshold value occurred in a machine check bank supporting CMCI (see Section 15.5.1, "CMCI Local APIC Interface").
- **LVT Timer Register (FEE0 0320H)** — Specifies interrupt delivery when the APIC timer signals an interrupt (see Section 10.5.4, "APIC Timer").
- **LVT Thermal Monitor Register (FEE0 0330H)** — Specifies interrupt delivery when the thermal sensor generates an interrupt (see Section 14.8.2, "Thermal Monitor"). This LVT entry is implementation specific, not architectural. If implemented, it will always be at base address FEE0 0330H.
- **LVT Performance Counter Register (FEE0 0340H)** — Specifies interrupt delivery when a performance counter generates an interrupt on overflow (see Section 19.6.3.5.8, "Generating an Interrupt on Overflow") or when Intel PT signals a ToPA PMI (see Section 32.2.7.2). This LVT entry is implementation specific, not architectural. If implemented, it is not guaranteed to be at base address FEE0 0340H.
- **LVT LINT0 Register (FEE0 0350H)** — Specifies interrupt delivery when an interrupt is signaled at the LINT0 pin.
- **LVT LINT1 Register (FEE0 0360H)** — Specifies interrupt delivery when an interrupt is signaled at the LINT1 pin.
- **LVT Error Register (FEE0 0370H)** — Specifies interrupt delivery when the APIC detects an internal error (see Section 10.5.3, "Error Handling").

The LVT performance counter register and its associated interrupt were introduced in the P6 processors and are also present in the Pentium 4 and Intel Xeon processors. The LVT thermal monitor register and its associated interrupt were introduced in the Pentium 4 and Intel Xeon processors. The LVT CMCI register and its associated interrupt were introduced in the Intel Xeon 5500 processors.

As shown in Figures 10-8, some of these fields and flags are not available (and reserved) for some entries.

The setup information that can be specified in the registers of the LVT table is as follows:

Vector Interrupt vector number.

Delivery Mode Specifies the type of interrupt to be sent to the processor. Some delivery modes will only operate as intended when used in conjunction with a specific trigger mode. The allowable delivery modes are as follows:

- **000 (Fixed)** Delivers the interrupt specified in the vector field.
- **010 (SMI)** Delivers an SMI interrupt to the processor core through the processor's local SMI signal path. When using this delivery mode, the vector field should be set to 00H for future compatibility.
- **100 (NMI)** Delivers an NMI interrupt to the processor. The vector information is ignored.
- **101 (INIT)** Delivers an INIT request to the processor core, which causes the processor to perform an INIT. When using this delivery mode, the vector field should be set to 00H for future compatibility. Not supported for the LVT CMCI register, the LVT thermal monitor register, or the LVT performance counter register.
- **110** Reserved; not supported for any LVT register.
- **111 (ExtINT)** Causes the processor to respond to the interrupt as if the interrupt originated in an externally connected (8259A-compatible) interrupt controller. A special INTA bus cycle corresponding to ExtINT, is routed to the external controller. The external controller is expected to supply the vector information. The APIC architecture supports only one ExtINT source in a system, usually contained in the compatibility bridge. Only one processor in the system should have an LVT entry configured to use the ExtINT delivery

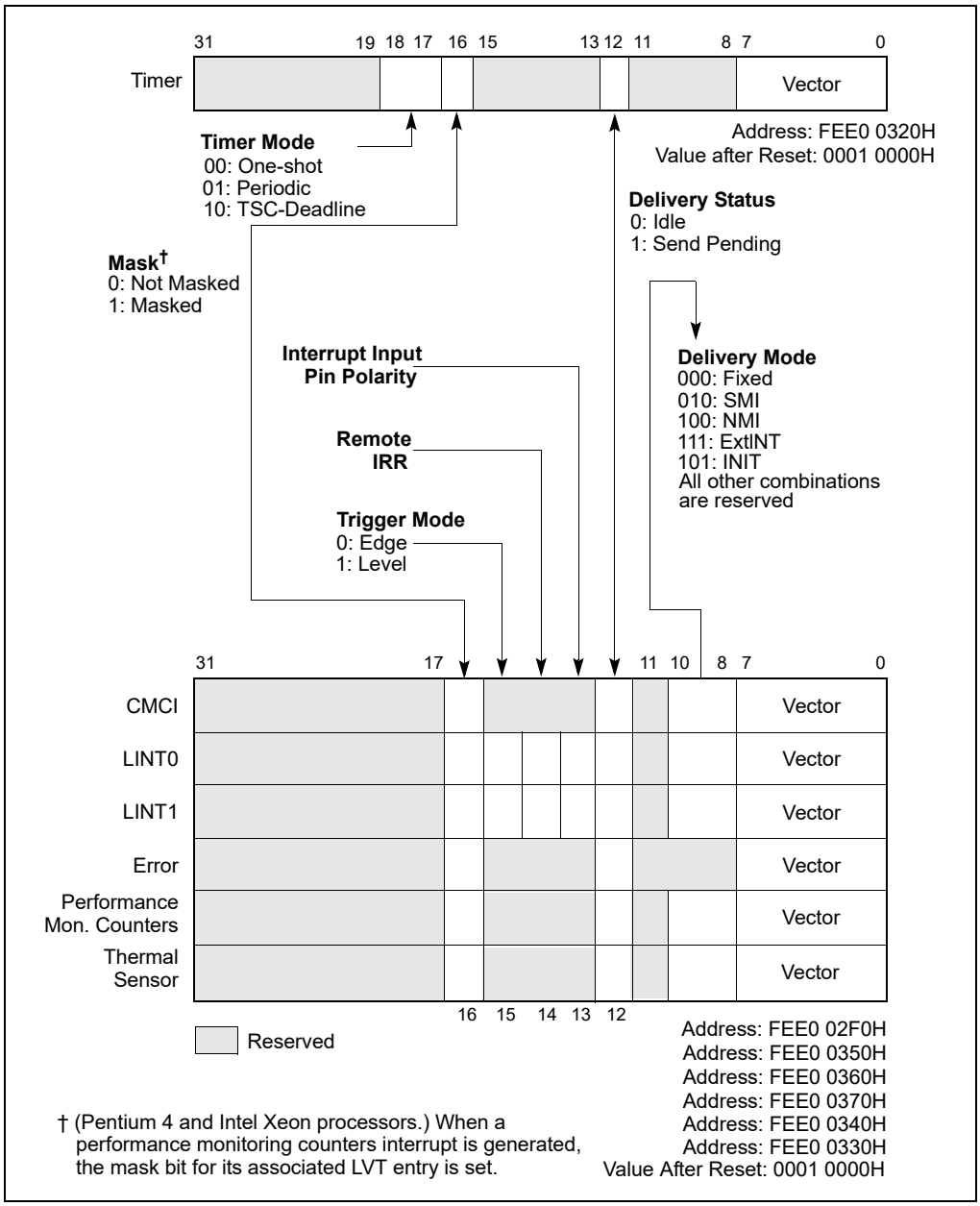


Figure 10-8. Local Vector Table (LVT)

mode. Not supported for the LVT CMCI register, the LVT thermal monitor register, or the LVT performance counter register.

Delivery Status (Read Only)

Indicates the interrupt delivery status, as follows:

0 (Idle) There is currently no activity for this interrupt source, or the previous interrupt from this source was delivered to the processor core and accepted.

1 (Send Pending)

Indicates that an interrupt from this source has been delivered to the processor core but has not yet been accepted (see Section 10.5.5, "Local Interrupt Acceptance").

Interrupt Input Pin Polarity
Specifies the polarity of the corresponding interrupt pin: (0) active high or (1) active low.

Remote IRR Flag (Read Only)
For fixed mode, level-triggered interrupts; this flag is set when the local APIC accepts the interrupt for servicing and is reset when an EOI command is received from the processor. The meaning of this flag is undefined for edge-triggered interrupts and other delivery modes.

Trigger Mode Selects the trigger mode for the local LINT0 and LINT1 pins: (0) edge sensitive and (1) level sensitive. This flag is only used when the delivery mode is Fixed. When the delivery mode is NMI, SMI, or INIT, the trigger mode is always edge sensitive. When the delivery mode is ExtINT, the trigger mode is always level sensitive. The timer and error interrupts are always treated as edge sensitive.

If the local APIC is not used in conjunction with an I/O APIC and fixed delivery mode is selected; the Pentium 4, Intel Xeon, and P6 family processors will always use level-sensitive triggering, regardless if edge-sensitive triggering is selected.

Software should always set the trigger mode in the LVT LINT1 register to 0 (edge sensitive). Level-sensitive interrupts are not supported for LINT1.

Mask Interrupt mask: (0) enables reception of the interrupt and (1) inhibits reception of the interrupt. When the local APIC handles a performance-monitoring counters interrupt, it automatically sets the mask flag in the LVT performance counter register. This flag is set to 1 on reset. It can be cleared only by software.

Timer Mode Bits 18:17 selects the timer mode (see Section 10.5.4):

(00b) one-shot mode using a count-down value,

(01b) periodic mode reloading a count-down value,

(10b) TSC-Deadline mode using absolute target value in IA32_TSC_DEADLINE MSR (see Section 10.5.4.1),

(11b) is reserved.

10.5.2 Valid Interrupt Vectors

The Intel 64 and IA-32 architectures define 256 vector numbers, ranging from 0 through 255 (see Section 6.2, "Exception and Interrupt Vectors"). Local and I/O APICs support 240 of these vectors (in the range of 16 to 255) as valid interrupts.

When an interrupt vector in the range of 0 to 15 is sent or received through the local APIC, the APIC indicates an illegal vector in its Error Status Register (see Section 10.5.3, "Error Handling"). The Intel 64 and IA-32 architectures reserve vectors 16 through 31 for predefined interrupts, exceptions, and Intel-reserved encodings (see Table 6-1). However, the local APIC does not treat vectors in this range as illegal.

When an illegal vector value (0 to 15) is written to an LVT entry and the delivery mode is Fixed (bits 8-11 equal 0), the APIC may signal an illegal vector error, without regard to whether the mask bit is set or whether an interrupt is actually seen on the input.

10.5.3 Error Handling

The local APIC records errors detected during interrupt handling in the error status register (ESR). The format of the ESR is given in Figure 10-9; it contains the following flags:

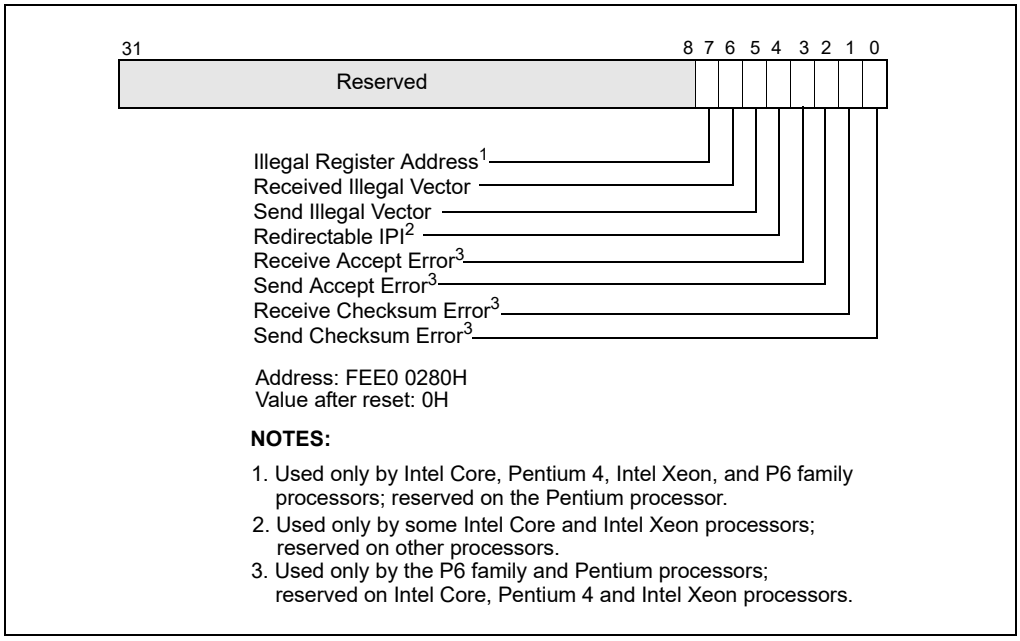


Figure 10-9. Error Status Register (ESR)

- Bit 0: Send Checksum Error.
  Set when the local APIC detects a checksum error for a message that it sent on the APIC bus. Used only on P6 family and Pentium processors.
- Bit 1: Receive Checksum Error.
  Set when the local APIC detects a checksum error for a message that it received on the APIC bus. Used only on P6 family and Pentium processors.
- Bit 2: Send Accept Error.
  Set when the local APIC detects that a message it sent was not accepted by any APIC on the APIC bus. Used only on P6 family and Pentium processors.
- Bit 3: Receive Accept Error.
  Set when the local APIC detects that the message it received was not accepted by any APIC on the APIC bus, including itself. Used only on P6 family and Pentium processors.

- Bit 4: Redirectable IPI.
  Set when the local APIC detects an attempt to send an IPI with the lowest-priority delivery mode and the local APIC does not support the sending of such IPIs. This bit is used on some Intel Core and Intel Xeon processors. As noted in Section 10.6.2, the ability of a processor to send a lowest-priority IPI is model-specific and should be avoided.
- Bit 5: Send Illegal Vector.
  Set when the local APIC detects an illegal vector (one in the range 0 to 15) in the message that it is sending. This occurs as the result of a write to the ICR (in both xAPIC and x2APIC modes) or to SELF IPI register (x2APIC mode only) with an illegal vector.

  If the local APIC does not support the sending of lowest-priority IPIs and software writes the ICR to send a lowest-priority IPI with an illegal vector, the local APIC sets only the "redirectable IPI" error bit. The interrupt is not processed and hence the "Send Illegal Vector" bit is not set in the ESR.
- Bit 6: Receive Illegal Vector.
  Set when the local APIC detects an illegal vector (one in the range 0 to 15) in an interrupt message it receives or in an interrupt generated locally from the local vector table or via a self IPI. Such interrupts are not delivered to the processor; the local APIC will never set an IRR bit in the range 0 to 15.
- Bit 7: Illegal Register Address
  Set when the local APIC is in xAPIC mode and software attempts to access a register that is reserved in the processor's local-APIC register-address space; see Table 10-1. (The local-APIC register-address space comprises the 4 KBytes at the physical address specified in the IA32_APIC_BASE MSR.) Used only on Intel Core, Intel Atom, Pentium 4, Intel Xeon, and P6 family processors.

  In x2APIC mode, software accesses the APIC registers using the RDMSR and WRMSR instructions. Use of one of these instructions to access a reserved register cause a general-protection exception (see Section 10.12.1.3). They do not set the "Illegal Register Access" bit in the ESR.

The ESR is a write/read register. Before attempt to read from the ESR, software should first write to it. (The value written does not affect the values read subsequently; only zero may be written in x2APIC mode.) This write clears any previously logged errors and updates the ESR with any errors detected since the last write to the ESR. This write also rearms the APIC error interrupt triggering mechanism.

The LVT Error Register (see Section 10.5.1) allows specification of the vector of the interrupt to be delivered to the processor core when APIC error is detected. The register also provides a means of masking an APIC-error interrupt. This masking only prevents delivery of APIC-error interrupts; the APIC continues to record errors in the ESR.

10.5.4 APIC Timer

The local APIC unit contains a 32-bit programmable timer that is available to software to time events or operations. This timer is set up by programming four registers: the divide configuration register (see Figure 10-10), the initial-count and current-count registers (see Figure 10-11), and the LVT timer register (see Figure 10-8).

If CPUID.06H:EAX.ARAT[bit 2] = 1, the processor's APIC timer runs at a constant rate regardless of P-state transitions and it continues to run at the same rate in deep C-states.

If CPUID.06H:EAX.ARAT[bit 2] = 0 or if CPUID 06H is not supported, the APIC timer may temporarily stop while the processor is in deep C-states or during transitions caused by Enhanced Intel SpeedStep® Technology.

The APIC timer frequency will be the processor's bus clock or core crystal clock frequency (when TSC/core crystal clock ratio is enumerated in CPUID leaf 0x15) divided by the value specified in the divide configuration register.

The timer can be configured through the timer LVT entry for one-shot or periodic operation. In one-shot mode, the timer is started by programming its initial-count register. The initial count value is then copied into the current-count register and count-down begins. After the timer reaches zero, a timer interrupt is generated and the timer remains at its 0 value until reprogrammed.

In periodic mode, the timer is started by writing to the initial-count register (as in one-shot mode), and the value written is copied into the current-count register, which counts down. The current-count register is automatically reloaded from the initial-count register when the count reaches 0 and a timer interrupt is generated, and the count-down is repeated. If during the count-down process the initial-count register is set, counting will restart, using the new initial-count value. The initial-count register is a read-write register; the current-count register is read only.

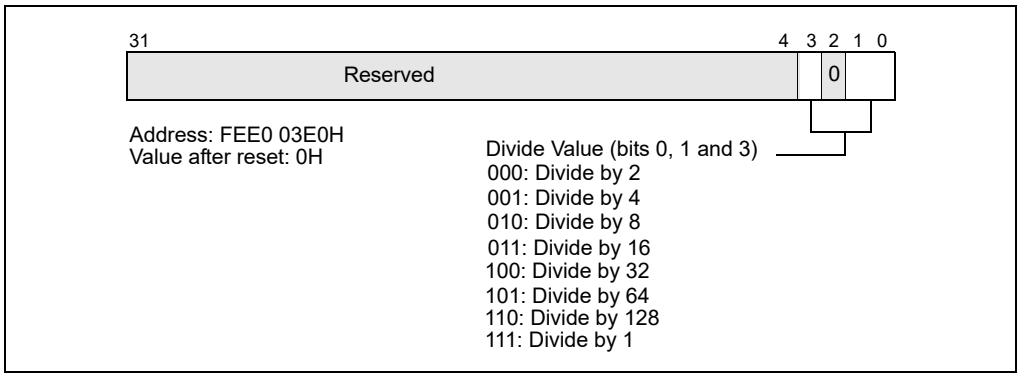


Figure 10-10. Divide Configuration Register

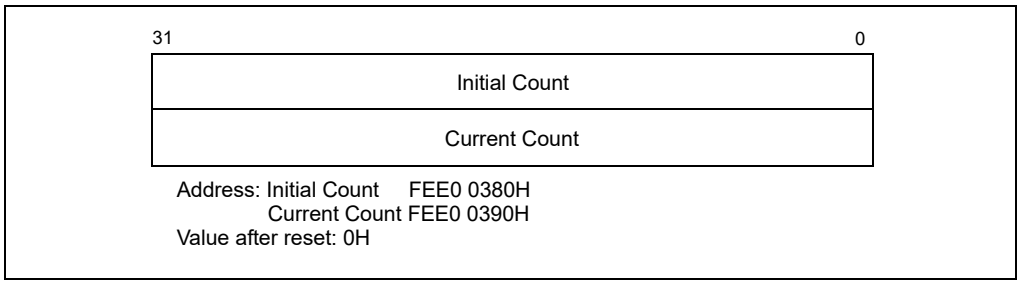


Figure 10-11. Initial Count and Current Count Registers

A write of 0 to the initial-count register effectively stops the local APIC timer, in both one-shot and periodic mode.

The LVT timer register determines the vector number that is delivered to the processor with the timer interrupt that is generated when the timer count reaches zero. The mask flag in the LVT timer register can be used to mask the timer interrupt.

NOTE

Changing the mode of the APIC timer (from one-shot to periodic or vice versa) by writing to the timer LVT entry does not start the timer. To start the timer, it is necessary to write to the initial-count register as described above.

10.5.4.1 TSC-Deadline Mode

The mode of operation of the local-APIC timer is determined by the LVT Timer Register. Specifically:

- If CPUID.01H:ECX.TSC_Deadline[bit 24] = 0, the mode is determined by bit 17 of the register.
- If CPUID.01H:ECX.TSC_Deadline[bit 24] = 1, the mode is determined by bits 18:17. See Figure 10-8. (If CPUID.01H:ECX.TSC_Deadline[bit 24] = 0, bit 18 of the register is reserved.)

The supported timer modes are given in Table 10-2. The three modes of the local APIC timer are mutually exclusive.

Table 10-2. Local APIC Timer Modes

LVT Bits [18:17]	Timer Mode
00b	One-shot mode, program count-down value in an initial-count register. See Section 10.5.4
01b	Periodic mode, program interval value in an initial-count register. See Section 10.5.4
10b	TSC-Deadline mode, program target value in IA32_TSC_DEADLINE MSR.
11b	Reserved

TSC-deadline mode allows software to use the local APIC timer to signal an interrupt at an absolute time. In TSC-deadline mode, writes to the initial-count register are ignored; and current-count register always reads 0. Instead, timer behavior is controlled using the IA32_TSC_DEADLINE MSR.

The IA32_TSC_DEADLINE MSR (MSR address 6E0H) is a per-logical processor MSR that specifies the time at which a timer interrupt should occur. Writing a non-zero 64-bit value into IA32_TSC_DEADLINE arms the timer. An interrupt is generated when the logical processor's time-stamp counter equals or exceeds the target value in the IA32_TSC_DEADLINE MSR.[1] When the timer generates an interrupt, it disarms itself and clears the IA32_TSC_DEADLINE MSR. Thus, each write to the IA32_TSC_DEADLINE MSR generates at most one timer interrupt.

In TSC-deadline mode, writing 0 to the IA32_TSC_DEADLINE MSR disarms the local-APIC timer. Transitioning between TSC-deadline mode and other timer modes also disarms the timer.

The hardware reset value of the IA32_TSC_DEADLINE MSR is 0. In other timer modes (LVT bit 18 = 0), the IA32_TSC_DEADLINE MSR reads zero and writes are ignored.

Software can configure the TSC-deadline timer to deliver a single interrupt using the following algorithm:

1. Detect support for TSC-deadline mode by verifying CPUID.1:ECX.24 = 1.
2. Select the TSC-deadline mode by programming bits 18:17 of the LVT Timer register with 10b.
3. Program the IA32_TSC_DEADLINE MSR with the target TSC value at which the timer interrupt is desired. This causes the processor to arm the timer.
4. The processor generates a timer interrupt when the value of time-stamp counter is greater than or equal to that of IA32_TSC_DEADLINE. It then disarms the timer and clear the IA32_TSC_DEADLINE MSR. (Both the time-stamp counter and the IA32_TSC_DEADLINE MSR are 64-bit unsigned integers.)
5. Software can re-arm the timer by repeating step 3.

The following are usage guidelines for TSC-deadline mode:

- Writes to the IA32_TSC_DEADLINE MSR are not serialized. Therefore, system software should not use WRMSR to the IA32_TSC_DEADLINE MSR as a serializing instruction. Read and write accesses to the IA32_TSC_DEADLINE and other MSR registers will occur in program order.
- Software can disarm the timer at any time by writing 0 to the IA32_TSC_DEADLINE MSR.
- If timer is armed, software can change the deadline (forward or backward) by writing a new value to the IA32_TSC_DEADLINE MSR.
- If software disarms the timer or postpones the deadline, race conditions may result in the delivery of a spurious timer interrupt. Software is expected to detect such spurious interrupts by checking the current value of the time-stamp counter to confirm that the interrupt was desired.[3]
- In xAPIC mode (in which the local-APIC registers are memory-mapped), software must order the memory-mapped write to the LVT entry that enables TSC-deadline mode and any subsequent WRMSR to the IA32_TSC_DEADLINE MSR. Software can assure proper ordering by executing the MFENCE instruction after the memory-mapped write and before any WRMSR. (In x2APIC mode, the WRMSR instruction is used to write to the LVT entry. The processor ensures the ordering of this write and any subsequent WRMSR to the deadline; no fencing is required.)

1. If the logical processor is in VMX non-root operation, a read of the time-stamp counter (using either RDMSR, RDTSC, or RDTSCP) may not return the actual value of the time-stamp counter; see Chapter 25 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*. It is the responsibility of software operating in VMX root operation to coordinate the virtualization of the time-stamp counter and the IA32_TSC_DEADLINE MSR.

10.5.5 Local Interrupt Acceptance

When a local interrupt is sent to the processor core, it is subject to the acceptance criteria specified in the interrupt acceptance flow chart in Figure 10-17. If the interrupt is accepted, it is logged into the IRR register and handled by the processor according to its priority (see Section 10.8.4, "Interrupt Acceptance for Fixed Interrupts"). If the interrupt is not accepted, it is sent back to the local APIC and retried.

10.6 ISSUING INTERPROCESSOR INTERRUPTS

The following sections describe the local APIC facilities that are provided for issuing interprocessor interrupts (IPIs) from software. The primary local APIC facility for issuing IPIs is the interrupt command register (ICR). The ICR can be used for the following functions:

- To send an interrupt to another processor.
- To allow a processor to forward an interrupt that it received but did not service to another processor for servicing.
- To direct the processor to interrupt itself (perform a self interrupt).
- To deliver special IPIs, such as the start-up IPI (SIPI) message, to other processors.

Interrupts generated with this facility are delivered to the other processors in the system through the system bus (for Pentium 4 and Intel Xeon processors) or the APIC bus (for P6 family and Pentium processors). The ability for a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software.

10.6.1 Interrupt Command Register (ICR)

The interrupt command register (ICR) is a 64-bit[1] local APIC register (see Figure 10-12) that allows software running on the processor to specify and send interprocessor interrupts (IPIs) to other processors in the system.

To send an IPI, software must set up the ICR to indicate the type of IPI message to be sent and the destination processor or processors. (All fields of the ICR are read-write by software with the exception of the delivery status field, which is read-only.) The act of writing to the low doubleword of the ICR causes the IPI to be sent.

1. In XAPIC mode the ICR is addressed as two 32-bit registers, ICR_LOW (FFE0 0300H) and ICR_HIGH (FFE0 0310H). In x2APIC mode, the ICR uses MSR 830H.

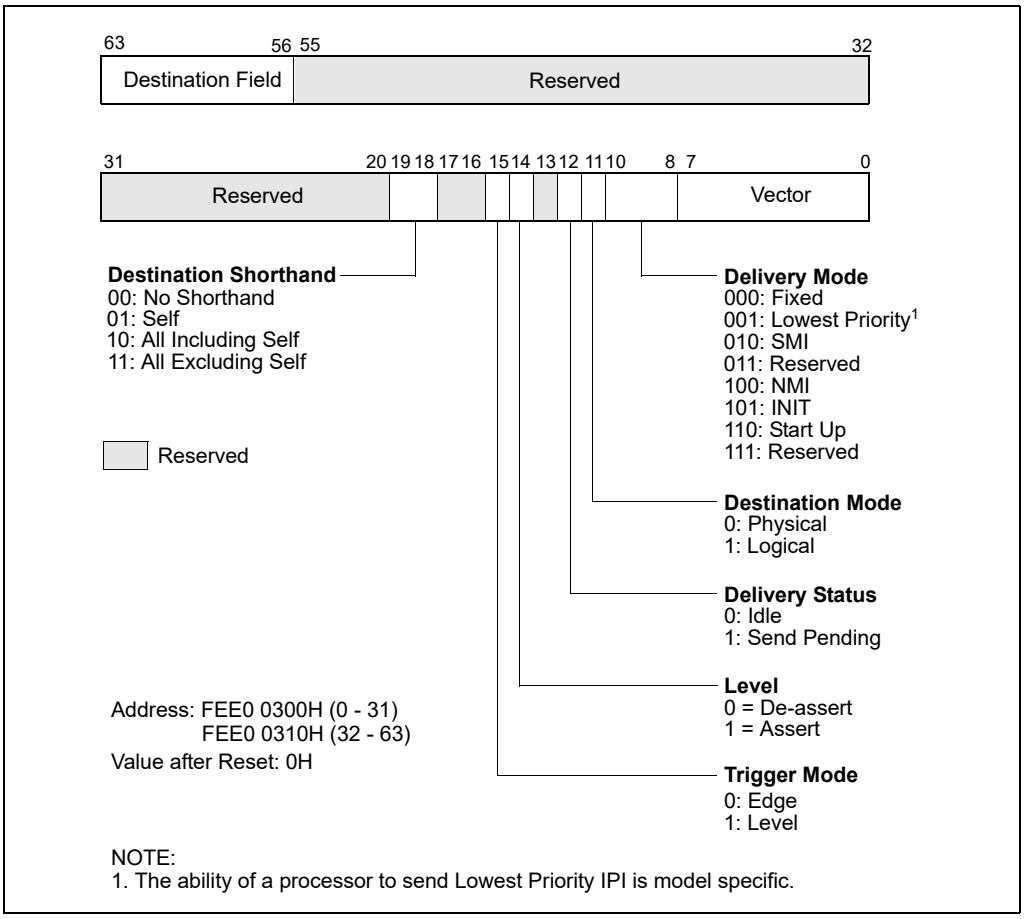


Figure 10-12. Interrupt Command Register (ICR)

The ICR consists of the following fields.

Vector The vector number of the interrupt being sent.

Delivery Mode Specifies the type of IPI to be sent. This field is also know as the IPI message type field.

- **000 (Fixed)** Delivers the interrupt specified in the vector field to the target processor or processors.
- **001 (Lowest Priority)** Same as fixed mode, except that the interrupt is delivered to the processor executing at the lowest priority among the set of processors specified in the destination field. The ability for a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software.
- **010 (SMI)** Delivers an SMI interrupt to the target processor or processors. The vector field must be programmed to 00H for future compatibility.
- **011 (Reserved)**
- **100 (NMI)** Delivers an NMI interrupt to the target processor or processors. The vector information is ignored.
- **101 (INIT)** Delivers an INIT request to the target processor or processors, which causes them to perform an INIT. As a result of this IPI message, all the target processors perform an INIT. The vector field must be programmed to 00H for future compatibility.
- **101 (INIT Level De-assert)** (Not supported in the Pentium 4 and Intel Xeon processors.) Sends a synchronization message to all the local APICs in the system to set their arbitration IDs (stored in their Arb ID registers) to the values of their APIC IDs (see Section 10.7, "System and APIC Bus Arbitration"). For this delivery mode, the level flag must be set to 0 and trigger mode flag to 1. This IPI is sent to all processors, regardless of the value in the destination field or the destination shorthand field; however, software should specify the "all including self" shorthand.
- **110 (Start-Up)** Sends a special "start-up" IPI (called a SIPI) to the target processor or processors. The vector typically points to a start-up routine that is part of the BIOS boot-strap code (see Section 8.4, "Multiple-Processor (MP) Initialization"). IPIs sent with this delivery mode are not automatically retried if the source APIC is unable to deliver it. It is up to the software to determine if the SIPI was not successfully delivered and to reissue the SIPI if necessary.

Destination Mode Selects either physical (0) or logical (1) destination mode (see Section 10.6.2, "Determining IPI Destination").

Delivery Status (Read Only) Indicates the IPI delivery status, as follows:

- **0 (Idle)** Indicates that this local APIC has completed sending any previous IPIs.
- **1 (Send Pending)** Indicates that this local APIC has not completed sending the last IPI.

Level For the INIT level de-assert delivery mode this flag must be set to 0; for all other delivery modes it must be set to 1. (This flag has no meaning in Pentium 4 and Intel Xeon processors, and will always be issued as a 1.)

Trigger Mode Selects the trigger mode when using the INIT level de-assert delivery mode: edge (0) or level (1). It is ignored for all other delivery modes. (This flag has no meaning in Pentium 4 and Intel Xeon processors, and will always be issued as a 0.)

Destination Shorthand

Indicates whether a shorthand notation is used to specify the destination of the interrupt and, if so, which shorthand is used. Destination shorthands are used in place of the 8-bit destination field, and can be sent by software using a single write to the low doubleword of the ICR. Shorthands are defined for the following cases: software self interrupt, IPIs to all processors in the system including the sender, IPIs to all processors in the system excluding the sender.

00: (No Shorthand)
The destination is specified in the destination field.

01: (Self) The issuing APIC is the one and only destination of the IPI. This destination shorthand allows software to interrupt the processor on which it is executing. An APIC implementation is free to deliver the self-interrupt message internally or to issue the message to the bus and "snoop" it as with any other IPI message.

10: (All Including Self)
The IPI is sent to all processors in the system including the processor sending the IPI. The APIC will broadcast an IPI message with the destination field set to FH for Pentium and P6 family processors and to FFH for Pentium 4 and Intel Xeon processors.

11: (All Excluding Self)
The IPI is sent to all processors in a system with the exception of the processor sending the IPI. The APIC broadcasts a message with the physical destination mode and destination field set to FH for Pentium and P6 family processors and to FFH for Pentium 4 and Intel Xeon processors. Support for this destination shorthand in conjunction with the lowest-priority delivery mode is model specific. For Pentium 4 and Intel Xeon processors, when this shorthand is used together with lowest priority delivery mode, the IPI may be redirected back to the issuing processor.

Destination Specifies the target processor or processors. This field is only used when the destination shorthand field is set to 00B. If the destination mode is set to physical, then bits 56 through 59 contain the APIC ID of the target processor for Pentium and P6 family processors and bits 56 through 63 contain the APIC ID of the target processor the for Pentium 4 and Intel Xeon processors. If the destination mode is set to logical, the interpretation of the 8-bit destination field depends on the settings of the DFR and LDR registers of the local APICs in all the processors in the system (see Section 10.6.2, "Determining IPI Destination").

Not all combinations of options for the ICR are valid. Table 10-3 shows the valid combinations for the fields in the ICR for the Pentium 4 and Intel Xeon processors; Table 10-4 shows the valid combinations for the fields in the ICR for the P6 family processors. Also note that the lower half of the ICR may not be preserved over transitions to the deepest C-States.

ICR operation in x2APIC mode is discussed in Section 10.12.9.

Table 10-3. Valid Combinations for Pentium 4 and Intel Xeon Processors Local xAPIC Interrupt Command Register

Destination Shorthand	Valid/Invalid	Trigger Mode	Delivery Mode	Destination Mode
No Shorthand	Valid	Edge	All Modes[1]	Physical or Logical
No Shorthand	Invalid[2]	Level	All Modes	Physical or Logical
Self	Valid	Edge	Fixed	X[3]
Self	Invalid[2]	Level	Fixed	X
Self	Invalid	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All Including Self	Valid	Edge	Fixed	X
All Including Self	Invalid[2]	Level	Fixed	X

Table 10-3. Valid Combinations for Pentium 4 and Intel Xeon Processors Local xAPIC Interrupt Command Register

Destination Shorthand	Valid/Invalid	Trigger Mode	Delivery Mode	Destination Mode
All Including Self	Invalid	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All Excluding Self	Valid	Edge	Fixed, Lowest Priority[1, 4], NMI, INIT, SMI, Start-Up	X
All Excluding Self	Invalid[2]	Level	FIxed, Lowest Priority[4], NMI, INIT, SMI, Start-Up	X

NOTES:

1. The ability of a processor to send a lowest priority IPI is model specific.
2. For these interrupts, if the trigger mode bit is 1 (Level), the local xAPIC will override the bit setting and issue the interrupt as an edge triggered interrupt.
3. X means the setting is ignored.
4. When using the "lowest priority" delivery mode and the "all excluding self" destination, the IPI can be redirected back to the issuing APIC, which is essentially the same as the "all including self" destination mode.

Table 10-4. Valid Combinations for the P6 Family Processor Local APIC Interrupt Command Register

Destination Shorthand	Valid/Invalid	Trigger Mode	Delivery Mode	Destination Mode
No Shorthand	Valid	Edge	All Modes[1]	Physical or Logical
No Shorthand	Valid[2]	Level	Fixed, Lowest Priority[1], NMI	Physical or Logical
No Shorthand	Valid[3]	Level	INIT	Physical or Logical
Self	Valid	Edge	Fixed	X[4]
Self	Valid[2]	Level	Fixed	X
Self	Invalid[5]	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All including Self	Valid	Edge	Fixed	X
All including Self	Valid[2]	Level	Fixed	X
All including Self	Invalid[5]	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All excluding Self	Valid	Edge	All Modes[1]	X
All excluding Self	Valid[2]	Level	Fixed, Lowest Priority[1], NMI	X
All excluding Self	Invalid[5]	Level	SMI, Start-Up	X
All excluding Self	Valid[3]	Level	INIT	X
X	Invalid[5]	Level	SMI, Start-Up	X

NOTES:

1. The ability of a processor to send a lowest priority IPI is model specific.
2. Treated as edge triggered if level bit is set to 1, otherwise ignored.
3. Treated as edge triggered when Level bit is set to 1; treated as "INIT Level Deassert" message when level bit is set to 0 (deassert). Only INIT level deassert messages are allowed to have the level bit set to 0. For all other messages the level bit must be set to 1.
4. X means the setting is ignored.
5. The behavior of the APIC is undefined.

10.6.2 Determining IPI Destination

The destination of an IPI[1] can be one, all, or a subset (group) of the processors on the system bus. The sender of the IPI specifies the destination of an IPI with the following APIC registers and fields within the registers:

- **ICR Register** — The following fields in the ICR register are used to specify the destination of an IPI.
  - **Destination Mode** — Selects one of two destination modes (physical or logical).
  - **Destination Field** — In physical destination mode, used to specify the APIC ID of the destination processor; in logical destination mode, used to specify a message destination address (MDA) that can be used to select specific processors in clusters.
  - **Destination Shorthand** — A quick method of specifying all processors, all excluding self, or self as the destination.
  - **Delivery mode, Lowest Priority** — Architecturally specifies that a lowest-priority arbitration mechanism be used to select a destination processor from a specified group of processors. The ability of a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software.
- **Local destination register (LDR)** — Used in conjunction with the logical destination mode and MDAs to select the destination processors.
- **Destination format register (DFR)** — Used in conjunction with the logical destination mode and MDAs to select the destination processors.

How the ICR, LDR, and DFR are used to select an IPI destination depends on the destination mode used: physical, logical, broadcast/self, or lowest-priority delivery mode. These destination modes are described in the following sections.

10.6.2.1 Physical Destination Mode

In physical destination mode, the destination processor is specified by its local APIC ID (see Section 10.4.6, "Local APIC ID"). For Pentium 4 and Intel Xeon processors, either a single destination (local APIC IDs 00H through FEH) or a broadcast to all APICs (the APIC ID is FFH) may be specified in physical destination mode.

A broadcast IPI (bits 28-31 of the MDA are 1's) or I/O subsystem initiated interrupt with lowest priority delivery mode is not supported in physical destination mode and must not be configured by software. Also, for any non-broadcast IPI or I/O subsystem initiated interrupt with lowest priority delivery mode, software must ensure that APICs defined in the interrupt address are present and enabled to receive interrupts.

For the P6 family and Pentium processors, a single destination is specified in physical destination mode with a local APIC ID of 0H through 0EH, allowing up to 15 local APICs to be addressed on the APIC bus. A broadcast to all local APICs is specified with 0FH.

NOTE

The number of local APICs that can be addressed on the system bus may be restricted by hardware.

10.6.2.2 Logical Destination Mode

In logical destination mode, IPI destination is specified using an 8-bit message destination address (MDA), which is entered in the destination field of the ICR. Upon receiving an IPI message that was sent using logical destination mode, a local APIC compares the MDA in the message with the values in its LDR and DFR to determine if it should accept and handle the IPI. For both configurations of logical destination mode, when combined with lowest priority delivery mode, software is responsible for ensuring that all of the local APICs included in or addressed by the IPI or I/O subsystem interrupt are present and enabled to receive the interrupt.

Figure 10-13 shows the layout of the logical destination register (LDR). The 8-bit logical APIC ID field in this register is used to create an identifier that can be compared with the MDA.

1. Determination of IPI destinations in x2APIC mode is discussed in Section 10.12.10.

NOTE

The logical APIC ID should not be confused with the local APIC ID that is contained in the local APIC ID register.

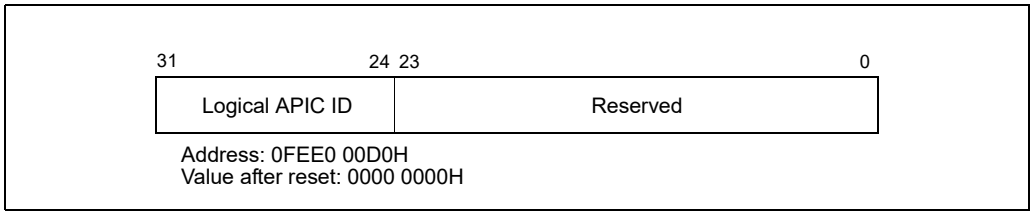


Figure 10-13. Logical Destination Register (LDR)

Figure 10-14 shows the layout of the destination format register (DFR). The 4-bit model field in this register selects one of two models (flat or cluster) that can be used to interpret the MDA when using logical destination mode.

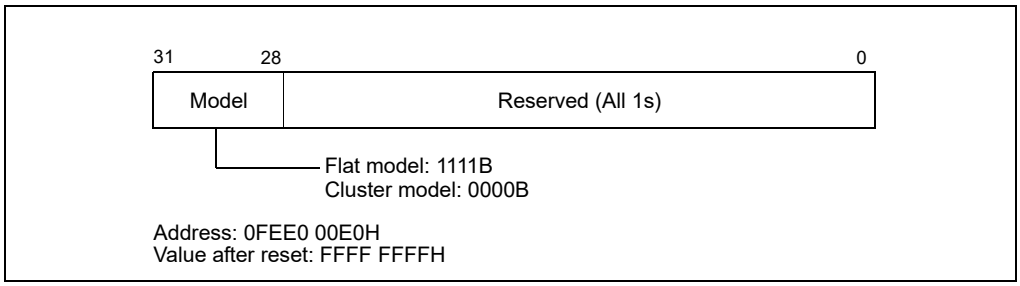


Figure 10-14. Destination Format Register (DFR)

The interpretation of MDA for the two models is described in the following paragraphs.

1. **Flat Model** — This model is selected by programming DFR bits 28 through 31 to 1111. Here, a unique logical APIC ID can be established for up to 8 local APICs by setting a different bit in the logical APIC ID field of the LDR for each local APIC. A group of local APICs can then be selected by setting one or more bits in the MDA.

   Each local APIC performs a bit-wise AND of the MDA and its logical APIC ID. If a true condition (non-zero) is detected, the local APIC accepts the IPI message. A broadcast to all APICs is achieved by setting the MDA to 1s.

2. **Cluster Model** — This model is selected by programming DFR bits 28 through 31 to 0000. This model supports two basic destination schemes: flat cluster and hierarchical cluster.

   The flat cluster destination model is only supported for P6 family and Pentium processors. Using this model, all APICs are assumed to be connected through the APIC bus. Bits 60 through 63 of the MDA contains the encoded address of the destination cluster and bits 56 through 59 identify up to four local APICs within the cluster (each bit is assigned to one local APIC in the cluster, as in the flat connection model). To identify one or more local APICs, bits 60 through 63 of the MDA are compared with bits 28 through 31 of the LDR to determine if a local APIC is part of the cluster. Bits 56 through 59 of the MDA are compared with Bits 24 through 27 of the LDR to identify a local APICs within the cluster.

   Sets of processors within a cluster can be specified by writing the target cluster address in bits 60 through 63 of the MDA and setting selected bits in bits 56 through 59 of the MDA, corresponding to the chosen members of the cluster. In this mode, 15 clusters (with cluster addresses of 0 through 14) each having 4 local APICs can be specified in the message. For the P6 and Pentium processor's local APICs, however, the APIC arbitration ID supports only 15 APIC agents. Therefore, the total number of processors and their local APICs supported in this mode is limited to 15. Broadcast to all local APICs is achieved by setting all destination bits to one. This guarantees a match on all clusters and selects all APICs in each cluster. A broadcast IPI or I/O subsystem broadcast interrupt with lowest priority delivery mode is not supported in cluster mode and must not be configured by software.

   The hierarchical cluster destination model can be used with Pentium 4, Intel Xeon, P6 family, or Pentium processors. With this model, a hierarchical network can be created by connecting different flat clusters via

independent system or APIC buses. This scheme requires a cluster manager within each cluster, which is responsible for handling message passing between system or APIC buses. One cluster contains up to 4 agents. Thus 15 cluster managers, each with 4 agents, can form a network of up to 60 APIC agents. Note that hierarchical APIC networks requires a special cluster manager device, which is not part of the local or the I/O APIC units.

NOTES

All processors that have their APIC software enabled (using the spurious vector enable/disable bit) must have their DFRs (Destination Format Registers) programmed identically.

The default mode for DFR is flat mode. If you are using cluster mode, DFRs must be programmed before the APIC is software enabled. Since some chipsets do not accurately track a system view of the logical mode, program DFRs as soon as possible after starting the processor.

10.6.2.3 Broadcast/Self Delivery Mode

The destination shorthand field of the ICR allows the delivery mode to be by-passed in favor of broadcasting the IPI to all the processors on the system bus and/or back to itself (see Section 10.6.1, "Interrupt Command Register (ICR)"). Three destination shorthands are supported: self, all excluding self, and all including self. The destination mode is ignored when a destination shorthand is used.

10.6.2.4 Lowest Priority Delivery Mode

With lowest priority delivery mode, the ICR is programmed to send an IPI to several processors on the system bus, using the logical or shorthand destination mechanism for selecting the processor. The selected processors then arbitrate with one another over the system bus or the APIC bus, with the lowest-priority processor accepting the IPI.

For systems based on the Intel Xeon processor, the chipset bus controller accepts messages from the I/O APIC agents in the system and directs interrupts to the processors on the system bus. When using the lowest priority delivery mode, the chipset chooses a target processor to receive the interrupt out of the set of possible targets. The Pentium 4 processor provides a special bus cycle on the system bus that informs the chipset of the current task priority for each logical processor in the system. The chipset saves this information and uses it to choose the lowest priority processor when an interrupt is received.

For systems based on P6 family processors, the processor priority used in lowest-priority arbitration is contained in the arbitration priority register (APR) in each local APIC. Figure 10-15 shows the layout of the APR.

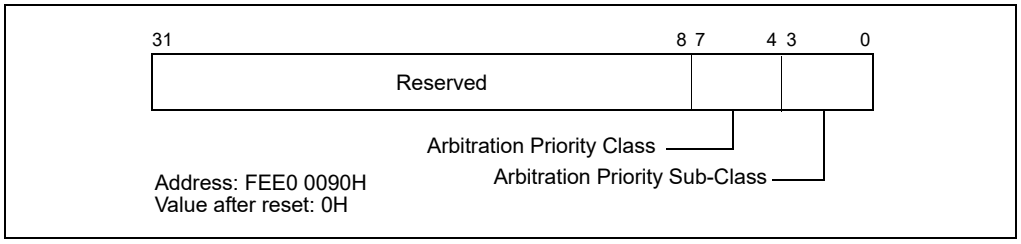


Figure 10-15. Arbitration Priority Register (APR)

The APR value is computed as follows:

```
IF (TPR[7:4] ≥ IRRV[7:4]) AND (TPR[7:4] > ISRV[7:4])
    THEN
        APR[7:0] ← TPR[7:0]
    ELSE
        APR[7:4] ← max(TPR[7:4] AND ISRV[7:4], IRRV[7:4])
        APR[3:0] ← 0.
```

Here, the TPR value is the task priority value in the TPR (see Figure 10-18), the IRRV value is the vector number for the highest priority bit that is set in the IRR (see Figure 10-20) or 00H (if no IRR bit is set), and the ISRV value is the vector number for the highest priority bit that is set in the ISR (see Figure 10-20). Following arbitration among the destination processors, the processor with the lowest value in its APR handles the IPI and the other processors ignore it.

(P6 family and Pentium processors.) For these processors, if a **focus processor** exists**,** it may accept the interrupt, regardless of its priority. A processor is said to be the focus of an interrupt if it is currently servicing that interrupt or if it has a pending request for that interrupt. For Intel Xeon processors, the concept of a focus processor is not supported.

In operating systems that use the lowest priority delivery mode but do not update the TPR, the TPR information saved in the chipset will potentially cause the interrupt to be always delivered to the same processor from the logical set. This behavior is functionally backward compatible with the P6 family processor but may result in unexpected performance implications.

10.6.3 IPI Delivery and Acceptance

When the low double-word of the ICR is written to, the local APIC creates an IPI message from the information contained in the ICR and sends the message out on the system bus (Pentium 4 and Intel Xeon processors) or the APIC bus (P6 family and Pentium processors). The manner in which these IPIs are handled after being issues in described in Section 10.8, "Handling Interrupts."

10.7 SYSTEM AND APIC BUS ARBITRATION

When several local APICs and the I/O APIC are sending IPI and interrupt messages on the system bus (or APIC bus), the order in which the messages are sent and handled is determined through bus arbitration.

For the Pentium 4 and Intel Xeon processors, the local and I/O APICs use the arbitration mechanism defined for the system bus to determine the order in which IPIs are handled. This mechanism is non-architectural and cannot be controlled by software.

For the P6 family and Pentium processors, the local and I/O APICs use an APIC-based arbitration mechanism to determine the order in which IPIs are handled. Here, each local APIC is given an arbitration priority of from 0 to 15, which the I/O APIC uses during arbitration to determine which local APIC should be given access to the APIC bus. The local APIC with the highest arbitration priority always wins bus access. Upon completion of an arbitration round, the winning local APIC lowers its arbitration priority to 0 and the losing local APICs each raise theirs by 1.

The current arbitration priority for a local APIC is stored in a 4-bit, software-transparent arbitration ID (Arb ID) register. During reset, this register is initialized to the APIC ID number (stored in the local APIC ID register). The INIT level-deassert IPI, which is issued with and ICR command, can be used to resynchronize the arbitration priorities of the local APICs by resetting Arb ID register of each agent to its current APIC ID value. (The Pentium 4 and Intel Xeon processors do not implement the Arb ID register.)

Section 10.10, "APIC Bus Message Passing Mechanism and Protocol (P6 Family, Pentium Processors)," describes the APIC bus arbitration protocols and bus message formats, while Section 10.6.1, "Interrupt Command Register (ICR)," describes the INIT level de-assert IPI message.

Note that except for the SIPI IPI (see Section 10.6.1, "Interrupt Command Register (ICR)"), all bus messages that fail to be delivered to their specified destination or destinations are automatically retried. Software should avoid situations in which IPIs are sent to disabled or nonexistent local APICs, causing the messages to be resent repeatedly. Additionally, interrupt sources that target the APIC should be masked or changed to no longer target the APIC.

10.8 HANDLING INTERRUPTS

When a local APIC receives an interrupt from a local source, an interrupt message from an I/O APIC, or an IPI, the manner in which it handles the message depends on processor implementation, as described in the following sections.

10.8.1 Interrupt Handling with the Pentium 4 and Intel Xeon Processors

With the Pentium 4 and Intel Xeon processors, the local APIC handles the local interrupts, interrupt messages, and IPIs it receives as follows:

1. It determines if it is the specified destination or not (see Figure 10-16). If it is the specified destination, it accepts the message; if it is not, it discards the message.

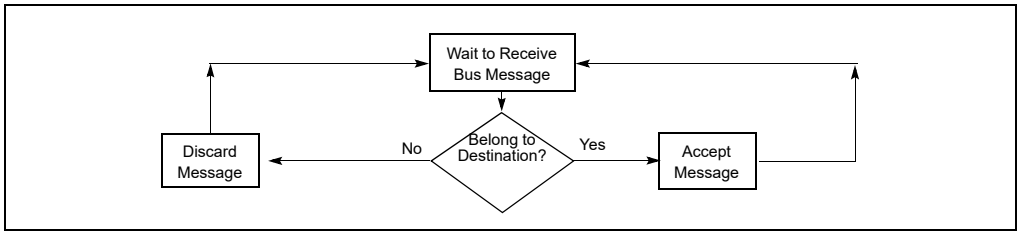


Figure 10-16. Interrupt Acceptance Flow Chart for the Local APIC (Pentium 4 and Intel Xeon Processors)

2. If the local APIC determines that it is the designated destination for the interrupt and if the interrupt request is an NMI, SMI, INIT, ExtINT, or SIPI, the interrupt is sent directly to the processor core for handling.
3. If the local APIC determines that it is the designated destination for the interrupt but the interrupt request is not one of the interrupts given in step 2, the local APIC sets the appropriate bit in the IRR.
4. When interrupts are pending in the IRR register, the local APIC dispatches them to the processor one at a time, based on their priority and the current processor priority in the PPR (see Section 10.8.3.1, "Task and Processor Priorities").
5. When a fixed interrupt has been dispatched to the processor core for handling, the completion of the handler routine is indicated with an instruction in the instruction handler code that writes to the end-of-interrupt (EOI) register in the local APIC (see Section 10.8.5, "Signaling Interrupt Servicing Completion"). The act of writing to the EOI register causes the local APIC to delete the interrupt from its ISR queue and (for level-triggered interrupts) send a message on the bus indicating that the interrupt handling has been completed. (A write to the EOI register must not be included in the handler routine for an NMI, SMI, INIT, ExtINT, or SIPI.)

10.8.2 Interrupt Handling with the P6 Family and Pentium Processors

With the P6 family and Pentium processors, the local APIC handles the local interrupts, interrupt messages, and IPIs it receives as follows (see Figure 10-17).

1. (IPIs only) The local APIC examines the IPI message to determines if it is the specified destination for the IPI as described in Section 10.6.2, "Determining IPI Destination." If it is the specified destination, it continues its acceptance procedure; if it is not the destination, it discards the IPI message. When the message specifies lowest-priority delivery mode, the local APIC will arbitrate with the other processors that were designated as recipients of the IPI message (see Section 10.6.2.4, "Lowest Priority Delivery Mode").
2. If the local APIC determines that it is the designated destination for the interrupt and if the interrupt request is an NMI, SMI, INIT, ExtINT, or INIT-deassert interrupt, or one of the MP protocol IPI messages (BIPI, FIPI, and SIPI), the interrupt is sent directly to the processor core for handling.
3. If the local APIC determines that it is the designated destination for the interrupt but the interrupt request is not one of the interrupts given in step 2, the local APIC looks for an open slot in one of its two pending interrupt queues contained in the IRR and ISR registers (see Figure 10-20). If a slot is available (see Section 10.8.4, "Interrupt Acceptance for Fixed Interrupts"), places the interrupt in the slot. If a slot is not available, it rejects the interrupt request and sends it back to the sender with a retry message.
4. When interrupts are pending in the IRR register, the local APIC dispatches them to the processor one at a time, based on their priority and the current processor priority in the PPR (see Section 10.8.3.1, "Task and Processor Priorities").
5. When a fixed interrupt has been dispatched to the processor core for handling, the completion of the handler routine is indicated with an instruction in the instruction handler code that writes to the end-of-interrupt (EOI)

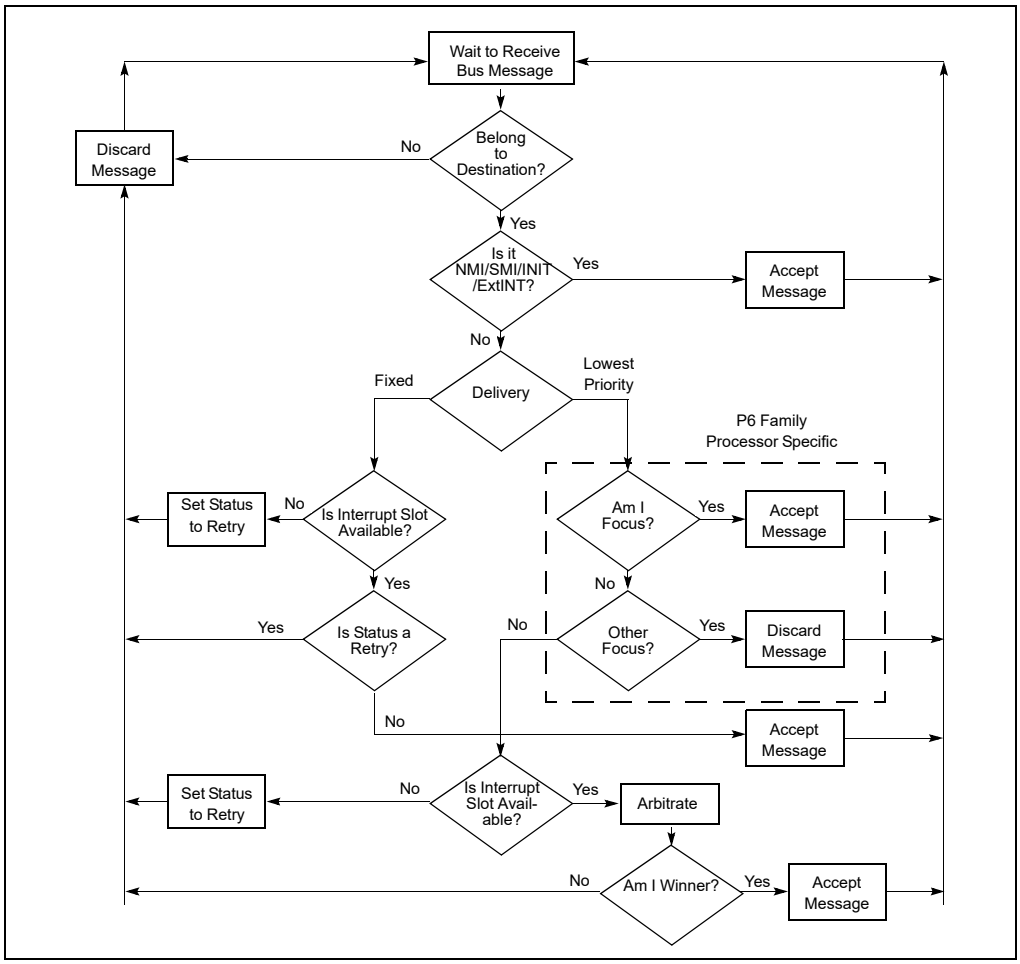


Figure 10-17. Interrupt Acceptance Flow Chart for the Local APIC (P6 Family and Pentium Processors)

register in the local APIC (see Section 10.8.5, "Signaling Interrupt Servicing Completion"). The act of writing to the EOI register causes the local APIC to delete the interrupt from its queue and (for level-triggered interrupts) send a message on the bus indicating that the interrupt handling has been completed. (A write to the EOI register must not be included in the handler routine for an NMI, SMI, INIT, ExtINT, or SIPI.)

The following sections describe the acceptance of interrupts and their handling by the local APIC and processor in greater detail.

10.8.3 Interrupt, Task, and Processor Priority

Each interrupt delivered to the processor through the local APIC has a priority based on its vector number. The local APIC uses this priority to determine when to service the interrupt relative to the other activities of the processor, including the servicing of other interrupts.

Each interrupt vector is an 8-bit value. The **interrupt-priority class** is the value of bits 7:4 of the interrupt vector. The lowest interrupt-priority class is 1 and the highest is 15; interrupts with vectors in the range 0–15 (with interrupt-priority class 0) are illegal and are never delivered. Because vectors 0–31 are reserved for dedicated uses by the Intel 64 and IA-32 architectures, software should configure interrupt vectors to use interrupt-priority classes in the range 2–15.

Each interrupt-priority class encompasses 16 vectors. The relative priority of interrupts within an interrupt-priority class is determined by the value of bits 3:0 of the vector number. The higher the value of those bits, the higher the

priority within that interrupt-priority class. Thus, each interrupt vector comprises two parts, with the high 4 bits indicating its interrupt-priority class and the low 4 bits indicating its ranking within the interrupt-priority class.

10.8.3.1 Task and Processor Priorities

The local APIC also defines a **task priority** and a **processor priority** that determine the order in which interrupts are handled. The **task-priority class** is the value of bits 7:4 of the task-priority register (TPR), which can be written by software (TPR is a read/write register); see Figure 10-18.

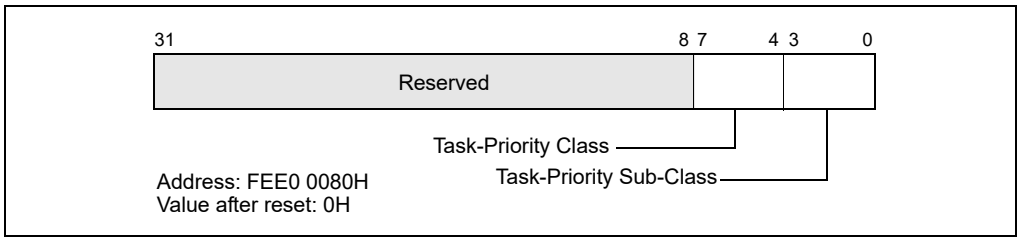


Figure 10-18. Task-Priority Register (TPR)

NOTE

In this discussion, the term "task" refers to a software defined task, process, thread, program, or routine that is dispatched to run on the processor by the operating system. It does not refer to an IA-32 architecture defined task as described in Chapter 7, "Task Management."

The task priority allows software to set a priority threshold for interrupting the processor. This mechanism enables the operating system to temporarily block low priority interrupts from disturbing high-priority work that the processor is doing. The ability to block such interrupts using task priority results from the way that the TPR controls the value of the processor-priority register (PPR).[1]

The **processor-priority class** is a value in the range 0–15 that is maintained in bits 7:4 of the processor-priority register (PPR); see Figure 10-19. The PPR is a read-only register. The processor-priority class represents the current priority at which the processor is executing.

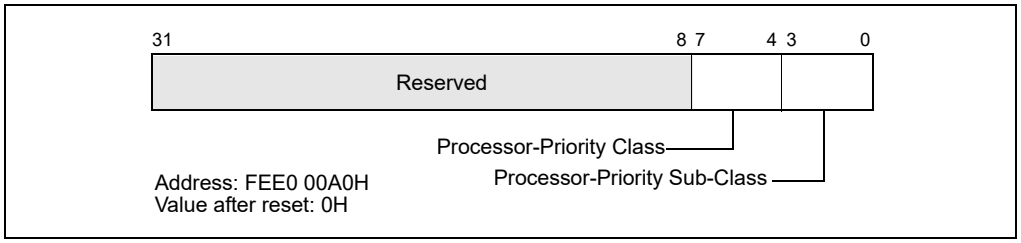


Figure 10-19. Processor-Priority Register (PPR)

The value of the PPR is based on the value of TPR and the value ISRV; ISRV is the vector number of the highest priority bit that is set in the ISR or 00H if no bit is set in the ISR. (See Section 10.8.4 for more details on the ISR.) The value of PPR is determined as follows:

- PPR[7:4] (the processor-priority class) the maximum of TPR[7:4] (the task- priority class) and ISRV[7:4] (the priority of the highest priority interrupt in service).
- PPR[3:0] (the processor-priority sub-class) is determined as follows:
  - If TPR[7:4] > ISRV[7:4], PPR[3:0] is TPR[3:0] (the task-priority sub-class).
  - If TPR[7:4] < ISRV[7:4], PPR[3:0] is 0.
  - If TPR[7:4] = ISRV[7:4], PPR[3:0] may be either TPR[3:0] or 0. The actual behavior is model-specific.

1. The TPR also determines the arbitration priority of the local processor; see Section 10.6.2.4, "Lowest Priority Delivery Mode."

The processor-priority class determines the priority threshold for interrupting the processor. The processor will deliver only those interrupts that have an interrupt-priority class higher than the processor-priority class in the PPR. If the processor-priority class is 0, the PPR does not inhibit the delivery any interrupt; if it is 15, the processor inhibits the delivery of all interrupts. (The processor-priority mechanism does not affect the delivery of interrupts with the NMI, SMI, INIT, ExtINT, INIT-deassert, and start-up delivery modes.)

The processor does not use the processor-priority sub-class to determine which interrupts to delivery and which to inhibit. (The processor uses the processor-priority sub-class only to satisfy reads of the PPR.)

10.8.4 Interrupt Acceptance for Fixed Interrupts

The local APIC queues the fixed interrupts that it accepts in one of two interrupt pending registers: the interrupt request register (IRR) or in-service register (ISR). These two 256-bit read-only registers are shown in Figure 10-20. The 256 bits in these registers represent the 256 possible vectors; vectors 0 through 15 are reserved by the APIC (see also: Section 10.5.2, "Valid Interrupt Vectors").

NOTE

All interrupts with an NMI, SMI, INIT, ExtINT, start-up, or INIT-deassert delivery mode bypass the IRR and ISR registers and are sent directly to the processor core for servicing.

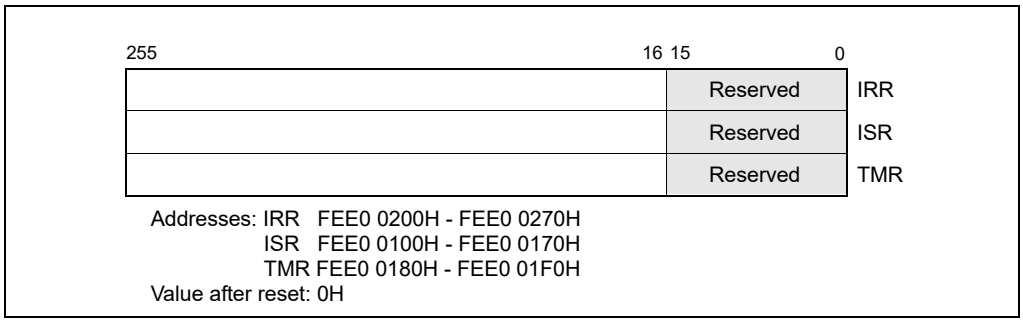


Figure 10-20. IRR, ISR and TMR Registers

The IRR contains the active interrupt requests that have been accepted, but not yet dispatched to the processor for servicing. When the local APIC accepts an interrupt, it sets the bit in the IRR that corresponds the vector of the accepted interrupt. When the processor core is ready to handle the next interrupt, the local APIC clears the highest priority IRR bit that is set and sets the corresponding ISR bit. The vector for the highest priority bit set in the ISR is then dispatched to the processor core for servicing.

While the processor is servicing the highest priority interrupt, the local APIC can send additional fixed interrupts by setting bits in the IRR. When the interrupt service routine issues a write to the EOI register (see Section 10.8.5, "Signaling Interrupt Servicing Completion"), the local APIC responds by clearing the highest priority ISR bit that is set. It then repeats the process of clearing the highest priority bit in the IRR and setting the corresponding bit in the ISR. The processor core then begins executing the service routing for the highest priority bit set in the ISR.

If more than one interrupt is generated with the same vector number, the local APIC can set the bit for the vector both in the IRR and the ISR. This means that for the Pentium 4 and Intel Xeon processors, the IRR and ISR can queue two interrupts for each interrupt vector: one in the IRR and one in the ISR. Any additional interrupts issued for the same interrupt vector are collapsed into the single bit in the IRR.

For the P6 family and Pentium processors, the IRR and ISR registers can queue no more than two interrupts per interrupt vector and will reject other interrupts that are received within the same vector.

If the local APIC receives an interrupt with an interrupt-priority class higher than that of the interrupt currently in service, and interrupts are enabled in the processor core, the local APIC dispatches the higher priority interrupt to the processor immediately (without waiting for a write to the EOI register). The currently executing interrupt handler is then interrupted so the higher-priority interrupt can be handled. When the handling of the higher-priority interrupt has been completed, the servicing of the interrupted interrupt is resumed.

The trigger mode register (TMR) indicates the trigger mode of the interrupt (see Figure 10-20). Upon acceptance of an interrupt into the IRR, the corresponding TMR bit is cleared for edge-triggered interrupts and set for level-triggered interrupts. If a TMR bit is set when an EOI cycle for its corresponding interrupt vector is generated, an EOI message is sent to all I/O APICs.

10.8.5 Signaling Interrupt Servicing Completion

For all interrupts except those delivered with the NMI, SMI, INIT, ExtINT, the start-up, or INIT-Deassert delivery mode, the interrupt handler must include a write to the end-of-interrupt (EOI) register (see Figure 10-21). This write must occur at the end of the handler routine, sometime before the IRET instruction. This action indicates that the servicing of the current interrupt is complete and the local APIC can issue the next interrupt from the ISR.

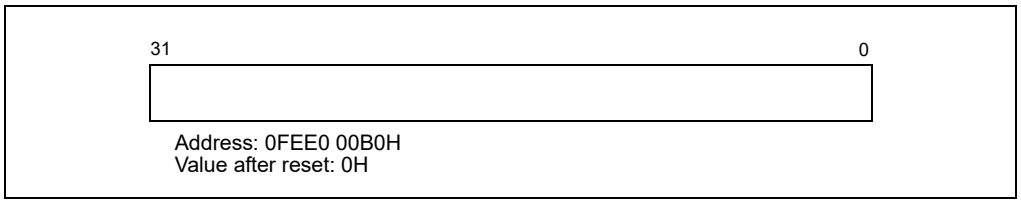


Figure 10-21. EOI Register

Upon receiving an EOI, the APIC clears the highest priority bit in the ISR and dispatches the next highest priority interrupt to the processor. If the terminated interrupt was a level-triggered interrupt, the local APIC also sends an end-of-interrupt message to all I/O APICs.

System software may prefer to direct EOIs to specific I/O APICs rather than having the local APIC send end-of-interrupt messages to all I/O APICs.

Software can inhibit the broadcast of EOI message by setting bit 12 of the Spurious Interrupt Vector Register (see Section 10.9). If this bit is set, a broadcast EOI is not generated on an EOI cycle even if the associated TMR bit indicates that the current interrupt was level-triggered. The default value for the bit is 0, indicating that EOI broadcasts are performed.

Bit 12 of the Spurious Interrupt Vector Register is reserved to 0 if the processor does not support suppression of EOI broadcasts. Support for EOI-broadcast suppression is reported in bit 24 in the Local APIC Version Register (see Section 10.4.8); the feature is supported if that bit is set to 1. When supported, the feature is available in both xAPIC mode and x2APIC mode.

System software desiring to perform directed EOIs for level-triggered interrupts should set bit 12 of the Spurious Interrupt Vector Register and follow each the EOI to the local xAPIC for a level triggered interrupt with a directed EOI to the I/O APIC generating the interrupt (this is done by writing to the I/O APIC's EOI register). System software performing directed EOIs must retain a mapping associating level-triggered interrupts with the I/O APICs in the system.

10.8.6 Task Priority in IA-32e Mode

In IA-32e mode, operating systems can manage the 16 interrupt-priority classes (see Section 10.8.3, "Interrupt, Task, and Processor Priority") explicitly using the task priority register (TPR). Operating systems can use the TPR to temporarily block specific (low-priority) interrupts from interrupting a high-priority task. This is done by loading TPR with a value in which the task-priority class corresponds to the highest interrupt-priority class that is to be blocked. For example:

- Loading the TPR with a task-priority class of 8 (01000B) blocks all interrupts with an interrupt-priority class of 8 or less while allowing all interrupts with an interrupt-priority class of 9 or more to be recognized.
- Loading the TPR with a task-priority class of 0 enables all external interrupts.
- Loading the TPR with a task-priority class of 0FH (01111B) disables all external interrupts.

The TPR (shown in Figure 10-18) is cleared to 0 on reset. In 64-bit mode, software can read and write the TPR using an alternate interface, MOV CR8 instruction. The new task-priority class is established when the MOV CR8

instruction completes execution. Software does not need to force serialization after loading the TPR using MOV CR8.

Use of the MOV CRn instruction requires a privilege level of 0. Programs running at privilege level greater than 0 cannot read or write the TPR. An attempt to do so causes a general-protection exception. The TPR is abstracted from the interrupt controller (IC), which prioritizes and manages external interrupt delivery to the processor. The IC can be an external device, such as an APIC or 8259. Typically, the IC provides a priority mechanism similar or identical to the TPR. The IC, however, is considered implementation-dependent with the under-lying priority mechanisms subject to change. CR8, by contrast, is part of the Intel 64 architecture. Software can depend on this definition remaining unchanged.

Figure 10-22 shows the layout of CR8; only the low four bits are used. The remaining 60 bits are reserved and must be written with zeros. Failure to do this causes a general-protection exception.

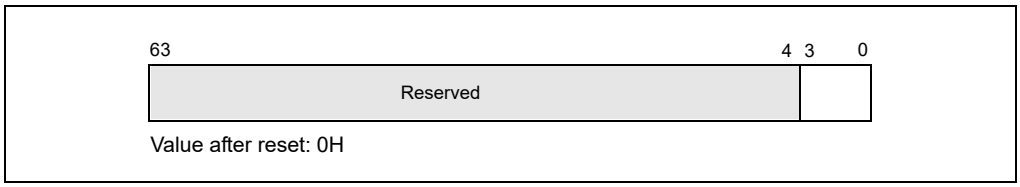


Figure 10-22. CR8 Register

10.8.6.1 Interaction of Task Priorities between CR8 and APIC

The first implementation of Intel 64 architecture includes a local advanced programmable interrupt controller (APIC) that is similar to the APIC used with previous IA-32 processors. Some aspects of the local APIC affect the operation of the architecturally defined task priority register and the programming interface using CR8.

Notable CR8 and APIC interactions are:

- The processor powers up with the local APIC enabled.
- The APIC must be enabled for CR8 to function as the TPR. Writes to CR8 are reflected into the APIC Task Priority Register.
- APIC.TPR[bits 7:4] = CR8[bits 3:0], APIC.TPR[bits 3:0] = 0. A read of CR8 returns a 64-bit value which is the value of TPR[bits 7:4], zero extended to 64 bits.

There are no ordering mechanisms between direct updates of the APIC.TPR and CR8. Operating software should implement either direct APIC TPR updates or CR8 style TPR updates but not mix them. Software can use a serializing instruction (for example, CPUID) to serialize updates between MOV CR8 and stores to the APIC.

10.9 SPURIOUS INTERRUPT

A special situation may occur when a processor raises its task priority to be greater than or equal to the level of the interrupt for which the processor INTR signal is currently being asserted. If at the time the INTA cycle is issued, the interrupt that was to be dispensed has become masked (programmed by software), the local APIC will deliver a spurious-interrupt vector. Dispensing the spurious-interrupt vector does not affect the ISR, so the handler for this vector should return without an EOI.

The vector number for the spurious-interrupt vector is specified in the spurious-interrupt vector register (see Figure 10-23). The functions of the fields in this register are as follows:

Spurious Vector Determines the vector number to be delivered to the processor when the local APIC generates a spurious vector.

(Pentium 4 and Intel Xeon processors.) Bits 0 through 7 of the this field are programmable by software.

(P6 family and Pentium processors). Bits 4 through 7 of the this field are programmable by software, and bits 0 through 3 are hardwired to logical ones. Software writes to bits 0 through 3 have no effect.

APIC Software Enable/Disable

Allows software to temporarily enable (1) or disable (0) the local APIC (see Section 10.4.3, "Enabling or Disabling the Local APIC").

Focus Processor Checking

Determines if focus processor checking is enabled (0) or disabled (1) when using the lowest-priority delivery mode. In Pentium 4 and Intel Xeon processors, this bit is reserved and should be cleared to 0.

Suppress EOI Broadcasts

Determines whether an EOI for a level-triggered interrupt causes EOI messages to be broadcast to the I/O APICs (0) or not (1). See Section 10.8.5. The default value for this bit is 0, indicating that EOI broadcasts are performed. This bit is reserved to 0 if the processor does not support EOI-broadcast suppression.

NOTE

Do not program an LVT or IOAPIC RTE with a spurious vector even if you set the mask bit. A spurious vector ISR does not do an EOI. If for some reason an interrupt is generated by an LVT or RTE entry, the bit in the in-service register will be left set for the spurious vector. This will mask all interrupts at the same or lower priority

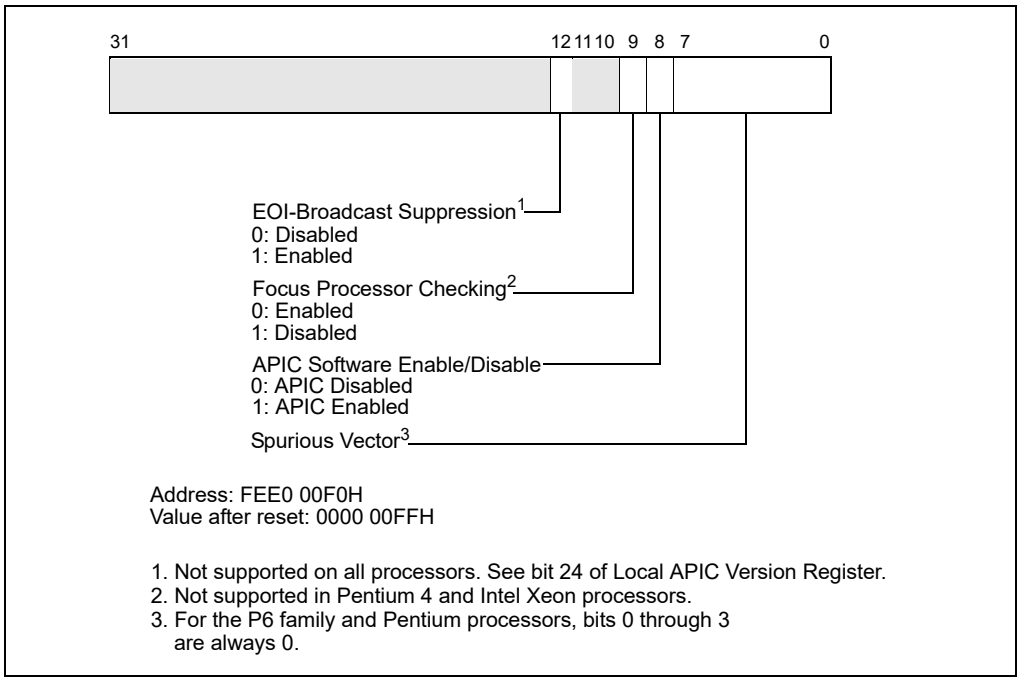


Figure 10-23. Spurious-Interrupt Vector Register (SVR)

10.10 APIC BUS MESSAGE PASSING MECHANISM AND PROTOCOL (P6 FAMILY, PENTIUM PROCESSORS)

The Pentium 4 and Intel Xeon processors pass messages among the local and I/O APICs on the system bus, using the system bus message passing mechanism and protocol.

The P6 family and Pentium processors, pass messages among the local and I/O APICs on the serial APIC bus, as follows. Because only one message can be sent at a time on the APIC bus, the I/O APIC and local APICs employ a "rotating priority" arbitration protocol to gain permission to send a message on the APIC bus. One or more APICs may start sending their messages simultaneously. At the beginning of every message, each APIC presents the type of the message it is sending and its current arbitration priority on the APIC bus. This information is used for arbitration. After each arbitration cycle (within an arbitration round), only the potential winners keep driving the bus.

By the time all arbitration cycles are completed, there will be only one APIC left driving the bus. Once a winner is selected, it is granted exclusive use of the bus, and will continue driving the bus to send its actual message.

After each successfully transmitted message, all APICs increase their arbitration priority by 1. The previous winner (that is, the one that has just successfully transmitted its message) assumes a priority of 0 (lowest). An agent whose arbitration priority was 15 (highest) during arbitration, but did not send a message, adopts the previous winner's arbitration priority, incremented by 1.

Note that the arbitration protocol described above is slightly different if one of the APICs issues a special End-Of-Interrupt (EOI). This high-priority message is granted the bus regardless of its sender's arbitration priority, unless more than one APIC issues an EOI message simultaneously. In the latter case, the APICs sending the EOI messages arbitrate using their arbitration priorities.

If the APICs are set up to use "lowest priority" arbitration (see Section 10.6.2.4, "Lowest Priority Delivery Mode") and multiple APICs are currently executing at the lowest priority (the value in the APR register), the arbitration priorities (unique values in the Arb ID register) are used to break ties. All 8 bits of the APR are used for the lowest priority arbitration.

10.10.1 Bus Message Formats

See Section 10.13, "APIC Bus Message Formats," for a description of bus message formats used to transmit messages on the serial APIC bus.

10.11 MESSAGE SIGNALLED INTERRUPTS

The *PCI Local Bus Specification, Rev 2.2* (www.pcisig.com) introduces the concept of message signalled interrupts. As the specification indicates:

> "Message signalled interrupts (MSI) is an optional feature that enables PCI devices to request service by writing a system-specified message to a system-specified address (PCI DWORD memory write transaction). The transaction address specifies the message destination while the transaction data specifies the message. System software is expected to initialize the message destination and message during device configuration, allocating one or more non-shared messages to each MSI capable function."

The capabilities mechanism provided by the *PCI Local Bus Specification* is used to identify and configure MSI capable PCI devices. Among other fields, this structure contains a Message Data Register and a Message Address Register. To request service, the PCI device function writes the contents of the Message Data Register to the address contained in the Message Address Register (and the Message Upper Address register for 64-bit message addresses).

Section 10.11.1 and Section 10.11.2 provide layout details for the Message Address Register and the Message Data Register. The operation issued by the device is a PCI write command to the Message Address Register with the Message Data Register contents. The operation follows semantic rules as defined for PCI write operations and is a DWORD operation.

10.11.1 Message Address Register Format

The format of the Message Address Register (lower 32-bits) is shown in Figure 10-24.

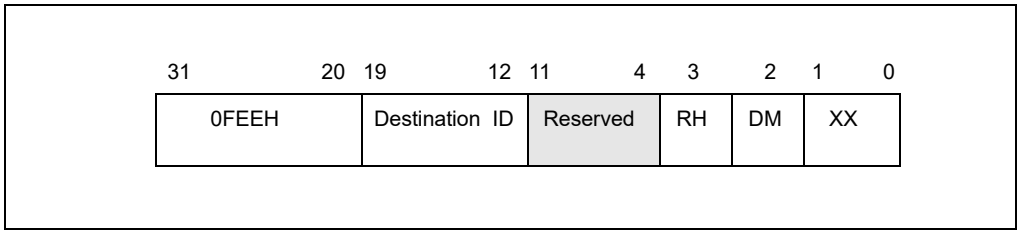


Figure 10-24. Layout of the MSI Message Address Register

Fields in the Message Address Register are as follows:

1. **Bits 31-20** — These bits contain a fixed value for interrupt messages (0FEEH). This value locates interrupts at the 1-MByte area with a base address of 4G – 18M. All accesses to this region are directed as interrupt messages. Care must to be taken to ensure that no other device claims the region as I/O space.
2. **Destination ID** — This field contains an 8-bit destination ID. It identifies the message's target processor(s). The destination ID corresponds to bits 63:56 of the I/O APIC Redirection Table Entry if the IOAPIC is used to dispatch the interrupt to the processor(s).
3. **Redirection hint indication** (RH) — When this bit is set, the message is directed to the processor with the lowest interrupt priority among processors that can receive the interrupt.
    - When RH is 0, the interrupt is directed to the processor listed in the Destination ID field.
    - When RH is 1 and the physical destination mode is used, the Destination ID field must not be set to FFH; it must point to a processor that is present and enabled to receive the interrupt.
    - When RH is 1 and the logical destination mode is active in a system using a flat addressing model, the Destination ID field must be set so that bits set to 1 identify processors that are present and enabled to receive the interrupt.
    - If RH is set to 1 and the logical destination mode is active in a system using cluster addressing model, then Destination ID field must not be set to FFH; the processors identified with this field must be present and enabled to receive the interrupt.
4. **Destination mode (DM)** — This bit indicates whether the Destination ID field should be interpreted as logical or physical APIC ID for delivery of the lowest priority interrupt.
    - If RH is 1 and DM is 0, the Destination ID field is in physical destination mode and only the processor in the system that has the matching APIC ID is considered for delivery of that interrupt (this means no re-direction).
    - If RH is 1 and DM is 1, the Destination ID Field is interpreted as in logical destination mode and the redirection is limited to only those processors that are part of the logical group of processors based on the processor's logical APIC ID and the Destination ID field in the message. The logical group of processors consists of those identified by matching the 8-bit Destination ID with the logical destination identified by the Destination Format Register and the Logical Destination Register in each local APIC. The details are similar to those described in Section 10.6.2, "Determining IPI Destination."
    - If RH is 0, then the DM bit is ignored and the message is sent ahead independent of whether the physical or logical destination mode is used.

10.11.2 Message Data Register Format

The layout of the Message Data Register is shown in Figure 10-25.

Reserved fields are not assumed to be any value. Software must preserve their contents on writes. Other fields in the Message Data Register are described below.

1. **Vector** — This 8-bit field contains the interrupt vector associated with the message. Values range from 010H to 0FEH. Software must guarantee that the field is not programmed with vector 00H to 0FH.
2. **Delivery Mode** — This 3-bit field specifies how the interrupt receipt is handled. Delivery Modes operate only in conjunction with specified Trigger Modes. Correct Trigger Modes must be guaranteed by software. Restrictions are indicated below:
    a. **000B (Fixed Mode)** — Deliver the signal to all the agents listed in the destination. The Trigger Mode for fixed delivery mode can be edge or level.
    b. **001B (Lowest Priority)** — Deliver the signal to the agent that is executing at the lowest priority of all agents listed in the destination field. The trigger mode can be edge or level.
    c. **010B (System Management Interrupt or SMI)** — The delivery mode is edge only. For systems that rely on SMI semantics, the vector field is ignored but must be programmed to all zeroes for future compatibility.

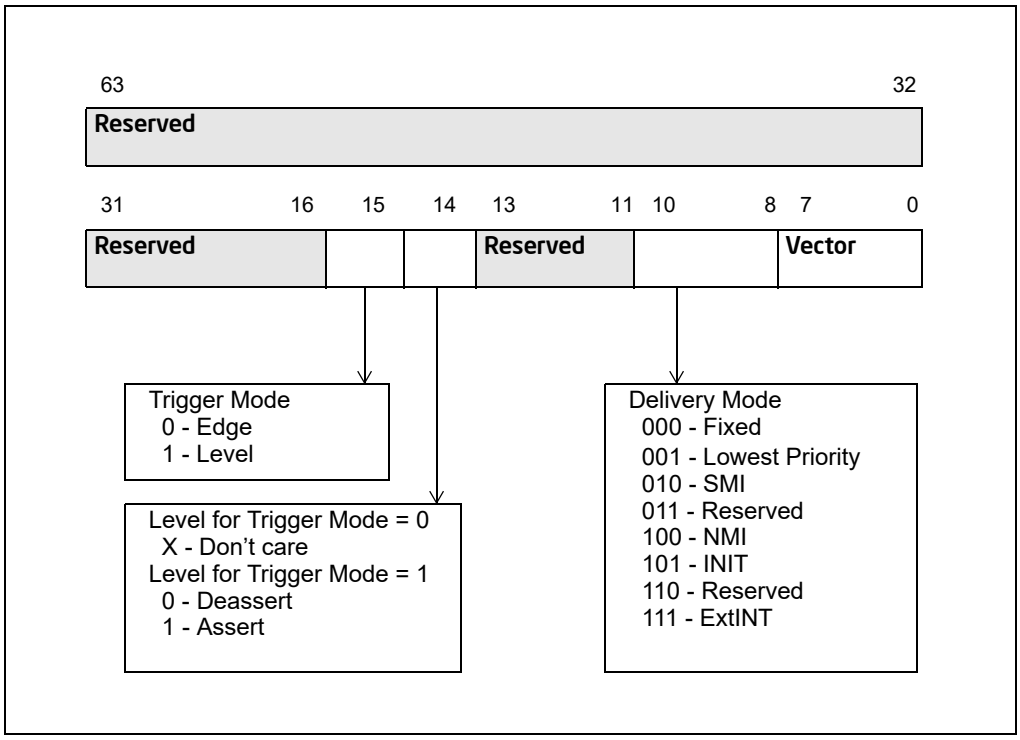


Figure 10-25. Layout of the MSI Message Data Register

d. **100B (NMI)** — Deliver the signal to all the agents listed in the destination field. The vector information is ignored. NMI is an edge triggered interrupt regardless of the Trigger Mode Setting.

e. **101B (INIT)** — Deliver this signal to all the agents listed in the destination field. The vector information is ignored. INIT is an edge triggered interrupt regardless of the Trigger Mode Setting.

f. **111B (ExtINT)** — Deliver the signal to the INTR signal of all agents in the destination field (as an interrupt that originated from an 8259A compatible interrupt controller). The vector is supplied by the INTA cycle issued by the activation of the ExtINT. ExtINT is an edge triggered interrupt.

3. **Level** — Edge triggered interrupt messages are always interpreted as assert messages. For edge triggered interrupts this field is not used. For level triggered interrupts, this bit reflects the state of the interrupt input.

4. **Trigger Mode** — This field indicates the signal type that will trigger a message.

   a. **0** — Indicates edge sensitive.

   b. **1** — Indicates level sensitive.

10.12 EXTENDED XAPIC (X2APIC)

The x2APIC architecture extends the xAPIC architecture (described in Section 10.4) in a backward compatible manner and provides forward extendability for future Intel platform innovations. Specifically, the x2APIC architecture does the following.

- Retains all key elements of compatibility to the xAPIC architecture.
  - — Delivery modes.
  - — Interrupt and processor priorities.
  - — Interrupt sources.
  - — Interrupt destination types.
- Provides extensions to scale processor addressability for both the logical and physical destination modes.

- Adds new features to enhance performance of interrupt delivery.
- Reduces complexity of logical destination mode interrupt delivery on link based platform architectures.
- Uses MSR programming interface to access APIC registers in x2APIC mode instead of memory-mapped interfaces. Memory-mapped interface is supported when operating in xAPIC mode.

10.12.1 Detecting and Enabling x2APIC Mode

Processor support for x2APIC mode can be detected by executing CPUID with EAX=1 and then checking ECX, bit 21 ECX. If CPUID.(EAX=1):ECX.21 is set , the processor supports the x2APIC capability and can be placed into the x2APIC mode.

System software can place the local APIC in the x2APIC mode by setting the x2APIC mode enable bit (bit 10) in the IA32_APIC_BASE MSR at MSR address 01BH. The layout for the IA32_APIC_BASE MSR is shown in Figure 10-26.

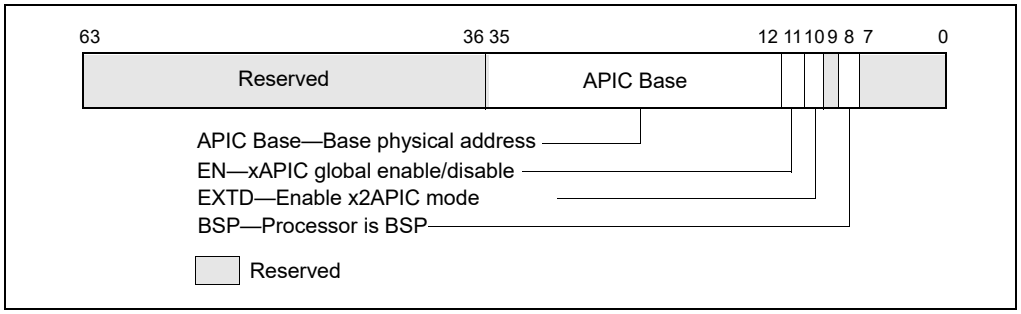


Figure 10-26. IA32_APIC_BASE MSR Supporting x2APIC

Table 10-5, "x2APIC operating mode configurations" describe the possible combinations of the enable bit (EN - bit 11) and the extended mode bit (EXTD - bit 10) in the IA32_APIC_BASE MSR.

Table 10-5. x2APIC Operating Mode Configurations

xAPIC global enable (IA32_APIC_BASE[11])	x2APIC enable (IA32_APIC_BASE[10])	Description
0	0	local APIC is disabled
0	1	Invalid
1	0	local APIC is enabled in xAPIC mode
1	1	local APIC is enabled in x2APIC mode

Once the local APIC has been switched to x2APIC mode (EN = 1, EXTD = 1), switching back to xAPIC mode would require system software to disable the local APIC unit. Specifically, attempting to write a value to the IA32_APIC_BASE MSR that has (EN= 1, EXTD = 0) when the local APIC is enabled and in x2APIC mode causes a general-protection exception. Once bit 10 in IA32_APIC_BASE MSR is set, the only way to leave x2APIC mode using IA32_APIC_BASE would require a WRMSR to set both bit 11 and bit 10 to zero. Section 10.12.5, "x2APIC State Transitions" provides a detailed state diagram for the state transitions allowed for the local APIC.

10.12.1.1 Instructions to Access APIC Registers

In x2APIC mode, system software uses RDMSR and WRMSR to access the APIC registers. The MSR addresses for accessing the x2APIC registers are architecturally defined and specified in Section 10.12.1.2, "x2APIC Register Address Space". Executing the RDMSR instruction with the APIC register address specified in ECX returns the content of bits 0 through 31 of the APIC registers in EAX. Bits 32 through 63 are returned in register EDX - these bits are reserved if the APIC register being read is a 32-bit register. Similarly executing the WRMSR instruction with the APIC register address in ECX, writes bits 0 to 31 of register EAX to bits 0 to 31 of the specified APIC register. If the register is a 64-bit register then bits 0 to 31 of register EDX are written to bits 32 to 63 of the APIC register. The

Interrupt Command Register is the only APIC register that is implemented as a 64-bit MSR. The semantics of handling reserved bits are defined in Section 10.12.1.3, "Reserved Bit Checking".

10.12.1.2 x2APIC Register Address Space

The MSR address range 800H through 8FFH is architecturally reserved and dedicated for accessing APIC registers in x2APIC mode. Table 10-6 lists the APIC registers that are available in x2APIC mode. When appropriate, the table also gives the offset at which each register is available on the page referenced by IA32_APIC_BASE[35:12] in xAPIC mode.

There is a one-to-one mapping between the x2APIC MSRs and the legacy xAPIC register offsets with the following exceptions:

- The Destination Format Register (DFR): The DFR, supported at offset 0E0H in xAPIC mode, is not supported in x2APIC mode. There is no MSR with address 80EH.
- The Interrupt Command Register (ICR): The two 32-bit registers in xAPIC mode (at offsets 300H and 310H) are merged into a single 64-bit MSR in x2APIC mode (with MSR address 830H). There is no MSR with address 831H.
- The SELF IPI register. This register is available only in x2APIC mode at address 83FH. In xAPIC mode, there is no register defined at offset 3F0H.

MSR addresses in the range 800H–8FFH that are not listed in Table 10-6 (including 80EH and 831H) are reserved. Executions of RDMSR and WRMSR that attempt to access such addresses cause general-protection exceptions.

The MSR address space is compressed to allow for future growth. Every 32 bit register on a 128-bit boundary in the legacy MMIO space is mapped to a single MSR in the local x2APIC MSR address space. The upper 32-bits of all x2APIC MSRs (except for the ICR) are reserved.

Table 10-6. Local APIC Register Address Map Supported by x2APIC

MSR Address (x2APIC mode)	MMIO Offset (xAPIC mode)	Register Name	MSR R/W Semantics	Comments
802H	020H	Local APIC ID register	Read-only[1]	See Section 10.12.5.1 for initial values.
803H	030H	Local APIC Version register	Read-only	Same version used in xAPIC mode and x2APIC mode.
808H	080H	Task Priority Register (TPR)	Read/write	Bits 31:8 are reserved.[2]
80AH	0A0H	Processor Priority Register (PPR)	Read-only	
80BH	0B0H	EOI register	Write-only[3]	WRMSR of a non-zero value causes #GP(0).
80DH	0D0H	Logical Destination Register (LDR)	Read-only	Read/write in xAPIC mode.
80FH	0F0H	Spurious Interrupt Vector Register (SVR)	Read/write	See Section 10.9 for reserved bits.
810H	100H	In-Service Register (ISR); bits 31:0	Read-only	
811H	110H	ISR bits 63:32	Read-only	
812H	120H	ISR bits 95:64	Read-only	
813H	130H	ISR bits 127:96	Read-only	
814H	140H	ISR bits 159:128	Read-only	
815H	150H	ISR bits 191:160	Read-only	
816H	160H	ISR bits 223:192	Read-only	

Table 10-6. Local APIC Register Address Map Supported by x2APIC (Contd.)

MSR Address (x2APIC mode)	MMIO Offset (xAPIC mode)	Register Name	MSR R/W Semantics	Comments
817H	170H	ISR bits 255:224	Read-only	
818H	180H	Trigger Mode Register (TMR); bits 31:0	Read-only	
819H	190H	TMR bits 63:32	Read-only	
81AH	1A0H	TMR bits 95:64	Read-only	
81BH	1B0H	TMR bits 127:96	Read-only	
81CH	1C0H	TMR bits 159:128	Read-only	
81DH	1D0H	TMR bits 191:160	Read-only	
81EH	1E0H	TMR bits 223:192	Read-only	
81FH	1F0H	TMR bits 255:224	Read-only	
820H	200H	Interrupt Request Register (IRR); bits 31:0	Read-only	
821H	210H	IRR bits 63:32	Read-only	
822H	220H	IRR bits 95:64	Read-only	
823H	230H	IRR bits 127:96	Read-only	
824H	240H	IRR bits 159:128	Read-only	
825H	250H	IRR bits 191:160	Read-only	
826H	260H	IRR bits 223:192	Read-only	
827H	270H	IRR bits 255:224	Read-only	
828H	280H	Error Status Register (ESR)	Read/write	WRMSR of a non-zero value causes #GP(0). See Section 10.5.3.
82FH	2F0H	LVT CMCI register	Read/write	See Figure 10-8 for reserved bits.
830H[4]	300H and 310H	Interrupt Command Register (ICR)	Read/write	See Figure 10-28 for reserved bits
832H	320H	LVT Timer register	Read/write	See Figure 10-8 for reserved bits.
833H	330H	LVT Thermal Sensor register	Read/write	See Figure 10-8 for reserved bits.
834H	340H	LVT Performance Monitoring register	Read/write	See Figure 10-8 for reserved bits.
835H	350H	LVT LINT0 register	Read/write	See Figure 10-8 for reserved bits.
836H	360H	LVT LINT1 register	Read/write	See Figure 10-8 for reserved bits.
837H	370H	LVT Error register	Read/write	See Figure 10-8 for reserved bits.
838H	380H	Initial Count register (for Timer)	Read/write	
839H	390H	Current Count register (for Timer)	Read-only	
83EH	3E0H	Divide Configuration Register (DCR; for Timer)	Read/write	See Figure 10-10 for reserved bits.
83FH	Not available	SELF IPI[5]	Write-only	Available only in x2APIC mode.

NOTES:

1. WRMSR causes #GP(0) for read-only registers.

2. WRMSR causes #GP(0) for attempts to set a reserved bit to 1 in a read/write register (including bits 63:32 of each register).
3. RDMSR causes #GP(0) for write-only registers.
4. MSR 831H is reserved; read/write operations cause general-protection exceptions. The contents of the APIC register at MMIO offset 310H are accessible in x2APIC mode through the MSR at address 830H.
5. SELF IPI register is supported only in x2APIC mode.

10.12.1.3 Reserved Bit Checking

Section 10.12.1.2 and Table 10-6 specifies the reserved bit definitions for the APIC registers in x2APIC mode. Non-zero writes (by WRMSR instruction) to reserved bits to these registers will raise a general protection fault exception while reads return zeros (RsvdZ semantics).

In x2APIC mode, the local APIC ID register is increased to 32 bits wide. This enables $2^{32}-1$ processors to be addressable in physical destination mode. This 32-bit value is referred to as "x2APIC ID". A processor implementation may choose to support less than 32 bits in its hardware. System software should be agnostic to the actual number of bits that are implemented. All non-implemented bits will return zeros on reads by software.

The APIC ID value of FFFF_FFFFH and the highest value corresponding to the implemented bit-width of the local APIC ID register in the system are reserved and cannot be assigned to any logical processor.

In x2APIC mode, the local APIC ID register is a read-only register to system software and will be initialized by hardware. It is accessed via the RDMSR instruction reading the MSR at address 0802H.

Each logical processor in the system (including clusters with a communication fabric) must be configured with an unique x2APIC ID to avoid collisions of x2APIC IDs. On DP and high-end MP processors targeted to specific market segments and depending on the system configuration, it is possible that logical processors in different and "unconnected" clusters power up initialized with overlapping x2APIC IDs. In these configurations, a model-specific means may be provided in those product segments to enable BIOS and/or platform firmware to re-configure the x2APIC IDs in some clusters to provide for unique and non-overlapping system wide IDs before configuring the disconnected components into a single system.

10.12.2 x2APIC Register Availability

The local APIC registers can be accessed via the MSR interface only when the local APIC has been switched to the x2APIC mode as described in Section 10.12.1. Accessing any APIC register in the MSR address range 0800H through 08FFH via RDMSR or WRMSR when the local APIC is not in x2APIC mode causes a general-protection exception. In x2APIC mode, the memory mapped interface is not available and any access to the MMIO interface will behave similar to that of a legacy xAPIC in globally disabled state. Table 10-7 provides the interactions between the legacy & extended modes and the legacy and register interfaces.

Table 10-7. MSR/MMIO Interface of a Local x2APIC in Different Modes of Operation

	MMIO Interface	MSR Interface
xAPIC mode	Available	General-protection exception
x2APIC mode	Behavior identical to xAPIC in globally disabled state	Available

10.12.3 MSR Access in x2APIC Mode

To allow for efficient access to the APIC registers in x2APIC mode, the serializing semantics of WRMSR are relaxed when writing to the APIC registers. Thus, system software should not use "WRMSR to APIC registers in x2APIC mode" as a serializing instruction. Read and write accesses to the APIC registers will occur in program order. A WRMSR to an APIC register may complete before all preceding stores are globally visible; software can prevent this by inserting a serializing instruction or the sequence MFENCE;LFENCE before the WRMSR.

The RDMSR instruction is not serializing and this behavior is unchanged when reading APIC registers in x2APIC mode. System software accessing the APIC registers using the RDMSR instruction should not expect a serializing behavior. (Note: The MMIO-based xAPIC interface is mapped by system software as an un-cached region. Consequently, read/writes to the xAPIC-MMIO interface have serializing semantics in the xAPIC mode.)

10.12.4 VM-Exit Controls for MSRs and x2APIC Registers

The VMX architecture allows a VMM to specify lists of MSRs to be loaded or stored on VMX transitions using the VMX-transition MSR areas (see VM-exit MSR-store address field, VM-exit MSR-load address field, and VM-entry MSR-load address field in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*).

The X2APIC MSRs cannot to be loaded and stored on VMX transitions. A VMX transition fails if the VMM has specified that the transition should access any MSRs in the address range from 0000_0800H to 0000_08FFH (the range used for accessing the X2APIC registers). Specifically, processing of an 128-bit entry in any of the VMX-transition MSR areas fails if bits 31:0 of that entry (represented as ENTRY_LOW_DW) satisfies the expression: "ENTRY_LOW_DW & FFFFF800H = 00000800H". Such a failure causes an associated VM entry to fail (by reloading host state) and causes an associated VM exit to lead to VMX abort.

10.12.5 x2APIC State Transitions

This section provides a detailed description of the x2APIC states of a local x2APIC unit, transitions between these states as well as interactions of these states with INIT and reset.

10.12.5.1 x2APIC States

The valid states for a local x2APIC unit are listed in Table 10-5.

- APIC disabled: IA32_APIC_BASE[EN]=0 and IA32_APIC_BASE[EXTD]=0.
- xAPIC mode: IA32_APIC_BASE[EN]=1 and IA32_APIC_BASE[EXTD]=0.
- x2APIC mode: IA32_APIC_BASE[EN]=1 and IA32_APIC_BASE[EXTD]=1.
- Invalid: IA32_APIC_BASE[EN]=0 and IA32_APIC_BASE[EXTD]=1.

The state corresponding to EXTD=1 and EN=0 is not valid and it is not possible to get into this state. An execution of WRMSR to the IA32_APIC_BASE_MSR that attempts a transition from a valid state to this invalid state causes a general-protection exception. Figure 10-27 shows the comprehensive state transition diagram for a local x2APIC unit.

On coming out of reset, the local APIC unit is enabled and is in the xAPIC mode: IA32_APIC_BASE[EN]=1 and IA32_APIC_BASE[EXTD]=0. The APIC registers are initialized as follows.

- The local APIC ID is initialized by hardware with a 32 bit ID (x2APIC ID). The lowest 8 bits of the x2APIC ID are the legacy local xAPIC ID, and are stored in the upper 8 bits of the APIC register for access in xAPIC mode.
- The following APIC registers are reset to all zeros for those fields that are defined in the xAPIC mode.
  - IRR, ISR, TMR, ICR, LDR, TPR, Divide Configuration Register (See Section 10.4 through Section 10.6 for details of individual APIC registers).
  - Timer initial count and timer current count registers.
- The LVT registers are reset to 0s except for the mask bits; these are set to 1s.
- The local APIC version register is not affected.
- The Spurious Interrupt Vector Register is initialized to 000000FFH.
- The DFR (available only in xAPIC mode) is reset to all 1s.
- SELF IPI register is reset to zero.

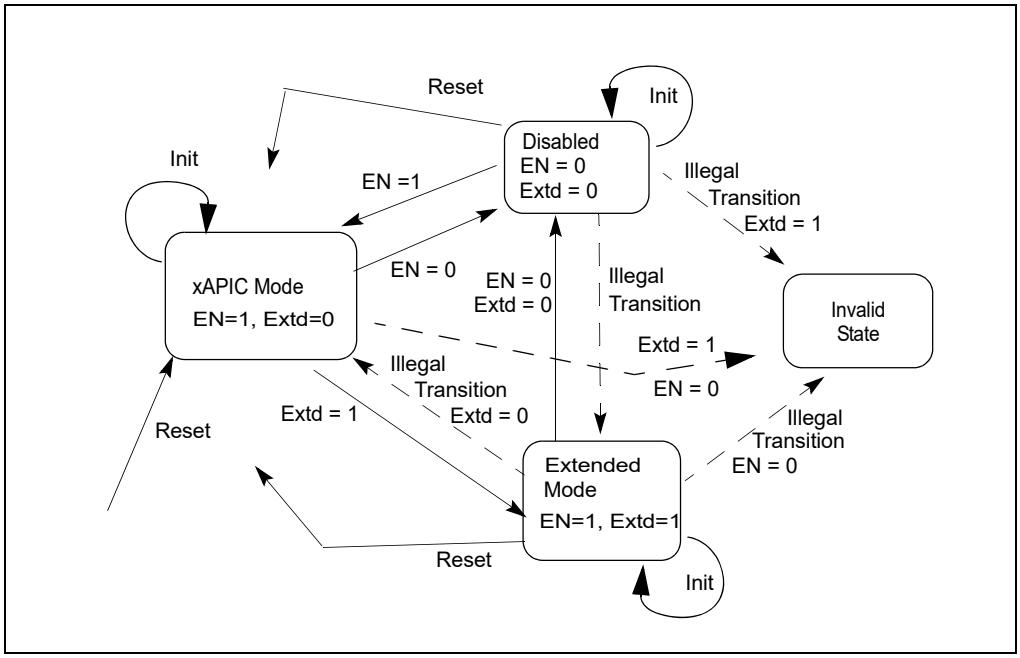


Figure 10-27. Local x2APIC State Transitions with IA32_APIC_BASE, INIT, and Reset

x2APIC After Reset

The valid transitions from the xAPIC mode state are:

- to the x2APIC mode by setting EXT to 1 (resulting EN=1, EXTD= 1). The physical x2APIC ID (see Figure 10-6) is preserved across this transition and the logical x2APIC ID (see Figure 10-29) is initialized by hardware during this transition as documented in Section 10.12.10.2. The state of the extended fields in other APIC registers, which was not initialized at reset, is not architecturally defined across this transition and system software should explicitly initialize those programmable APIC registers.
- to the disabled state by setting EN to 0 (resulting EN=0, EXTD= 0).

The result of an INIT in the xAPIC state places the APIC in the state with EN= 1, EXTD= 0. The state of the local APIC ID register is preserved (the 8-bit xAPIC ID is in the upper 8 bits of the APIC ID register). All the other APIC registers are initialized as a result of INIT.

A reset in this state places the APIC in the state with EN= 1, EXTD= 0. The state of the local APIC ID register is initialized as described in Section 10.12.5.1. All the other APIC registers are initialized described in Section 10.12.5.1.

x2APIC Transitions From x2APIC Mode

From the x2APIC mode, the only valid x2APIC transition using IA32_APIC_BASE is to the state where the x2APIC is disabled by setting EN to 0 and EXTD to 0. The x2APIC ID (32 bits) and the legacy local xAPIC ID (8 bits) are preserved across this transition. A transition from the x2APIC mode to xAPIC mode is not valid, and the corresponding WRMSR to the IA32_APIC_BASE MSR causes a general-protection exception.

A reset in this state places the x2APIC in xAPIC mode. All APIC registers (including the local APIC ID register) are initialized as described in Section 10.12.5.1.

An INIT in this state keeps the x2APIC in the x2APIC mode. The state of the local APIC ID register is preserved (all 32 bits). However, all the other APIC registers are initialized as a result of the INIT transition.

x2APIC Transitions From Disabled Mode

From the disabled state, the only valid x2APIC transition using IA32_APIC_BASE is to the xAPIC mode (EN= 1, EXTD = 0). Thus the only means to transition from x2APIC mode to xAPIC mode is a two-step process:

- first transition from x2APIC mode to local APIC disabled mode (EN= 0, EXTD = 0),
- followed by another transition from disabled mode to xAPIC mode (EN= 1, EXTD= 0).

Consequently, all the APIC register states in the x2APIC, except for the x2APIC ID (32 bits), are not preserved across mode transitions.

A reset in the disabled state places the x2APIC in the xAPIC mode. All APIC registers (including the local APIC ID register) are initialized as described in Section 10.12.5.1.

An INIT in the disabled state keeps the x2APIC in the disabled state.

State Changes From xAPIC Mode to x2APIC Mode

After APIC register states have been initialized by software in xAPIC mode, a transition from xAPIC mode to x2APIC mode does not affect most of the APIC register states, except the following:

- The Logical Destination Register is not preserved.
- Any APIC ID value written to the memory-mapped local APIC ID register is not preserved.
- The high half of the Interrupt Command Register is not preserved.

10.12.6 Routing of Device Interrupts in x2APIC Mode

The x2APIC architecture is intended to work with all existing IOxAPIC units as well as all PCI and PCI Express (PCIe) devices that support the capability for message-signaled interrupts (MSI). Support for x2APIC modifies only the following:

- the local APIC units;
- the interconnects joining IOxAPIC units to the local APIC units; and
- the interconnects joining MSI-capable PCI and PCIe devices to the local APIC units.

No modifications are required to MSI-capable PCI and PCIe devices. Similarly, no modifications are required to IOxAPIC units. This made possible through use of the interrupt-remapping architecture specified in the *Intel® Virtualization Technology for Directed I/O*, Revision 1.3 for the routing of interrupts from MSI-capable devices to local APIC units operating in x2APIC mode.

10.12.7 Initialization by System Software

Routing of device interrupts to local APIC units operating in x2APIC mode requires use of the interrupt-remapping architecture specified in the *Intel® Virtualization Technology for Directed I/O* (Revision 1.3 and/or later versions). Because of this, BIOS must enumerate support for and software must enable this interrupt remapping with Extended Interrupt Mode Enabled before it enabling x2APIC mode in the local APIC units.

The ACPI interfaces for the x2APIC are described in Section 5.2, "ACPI System Description Tables," of the *Advanced Configuration and Power Interface Specification*, Revision 4.0a (http://www.acpi.info/spec.htm). The default behavior for BIOS is to pass the control to the operating system with the local x2APICs in xAPIC mode if all APIC IDs reported by CPUID.0BH:EDX are less than 255, and in x2APIC mode if there are any logical processor reporting an APIC ID of 255 or greater.

10.12.8 CPUID Extensions And Topology Enumeration

For Intel 64 and IA-32 processors that support x2APIC, a value of 1 reported by CPUID.01H:ECX[21] indicates that the processor supports x2APIC and the extended topology enumeration leaf (CPUID.0BH).

The extended topology enumeration leaf can be accessed by executing CPUID with EAX = 0BH. Processors that do not support x2APIC may support CPUID leaf 0BH. Software can detect the availability of the extended topology enumeration leaf (0BH) by performing two steps:

- Check maximum input value for basic CPUID information by executing CPUID with EAX= 0. If CPUID.0H:EAX is greater than or equal or 11 (0BH), then proceed to next step
- Check CPUID.EAX=0BH, ECX=0H:EBX is non-zero.

If both of the above conditions are true, extended topology enumeration leaf is available. If available, the extended topology enumeration leaf is the preferred mechanism for enumerating topology. The presence of CPUID leaf 0BH in a processor does not guarantee support for x2APIC. If CPUID.EAX=0BH, ECX=0H: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.

The extended topology enumeration leaf is intended to assist software with enumerating processor topology on systems that requires 32-bit x2APIC IDs to address individual logical processors. Details of CPUID leaf 0BH can be found in the reference pages of CPUID in Chapter 3 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

Processor topology enumeration algorithm for processors supporting the extended topology enumeration leaf of CPUID and processors that do not support CPUID leaf 0BH are treated in Section 8.9.4, "Algorithm for Three-Level Mappings of APIC_ID".

10.12.8.1 Consistency of APIC IDs and CPUID

The consistency of physical x2APIC ID in MSR 802H in x2APIC mode and the 32-bit value returned in CPUID.0BH:EDX is facilitated by processor hardware.

CPUID.0BH:EDX will report the full 32 bit ID, in xAPIC and x2APIC mode. This allows BIOS to determine if a system has processors with IDs exceeding the 8-bit initial APIC ID limit (CPUID.01H:EBX[31:24]). Initial APIC ID (CPUID.01H:EBX[31:24]) is always equal to CPUID.0BH:EDX[7:0].

If the values of CPUID.0BH:EDX reported by all logical processors in a system are less than 255, BIOS can transfer control to OS in xAPIC mode.

If the values of CPUID.0BH:EDX reported by some logical processors in a system are greater than or equal to 255, BIOS must support two options to hand off to OS.

- If BIOS enables logical processors with x2APIC IDs greater than 255, then it should enable x2APIC in the Boot Strap Processor (BSP) and all Application Processors (AP) before passing control to the OS. Applications requiring processor topology information must use OS provided services based on x2APIC IDs or CPUID.0BH leaf.
- If a BIOS transfers control to OS in xAPIC mode, then the BIOS must ensure that only logical processors with CPUID.0BH.EDX value less than 255 are enabled. BIOS initialization on all logical processors with CPUID.0B.EDX values greater than or equal to 255 must (a) disable APIC and execute CLI in each logical processor, and (b) leave these logical processor in the lowest power state so that these processors do not respond to INIT IPI during OS boot. The BSP and all the enabled logical processor operate in xAPIC mode after BIOS passed control to OS. Application requiring processor topology information can use OS provided legacy services based on 8-bit initial APIC IDs or legacy topology information from CPUID.01H and CPUID 04H leaves. Even if the BIOS passes control in xAPIC mode, an OS can switch the processors to x2APIC mode later. BIOS SMM handler should always read the APIC_BASE_MSR, determine the APIC mode and use the corresponding access method.

10.12.9 ICR Operation in x2APIC Mode

In x2APIC mode, the layout of the Interrupt Command Register is shown in Figure 10-12. The lower 32 bits of ICR in x2APIC mode is identical to the lower half of the ICR in xAPIC mode, except the Delivery Status bit is removed since it is not needed in x2APIC mode. The destination ID field is expanded to 32 bits in x2APIC mode.

To send an IPI using the ICR, software must set up the ICR to indicate the type of IPI message to be sent and the destination processor or processors. Self IPIs can also be sent using the SELF IPI register (see Section 10.12.11).

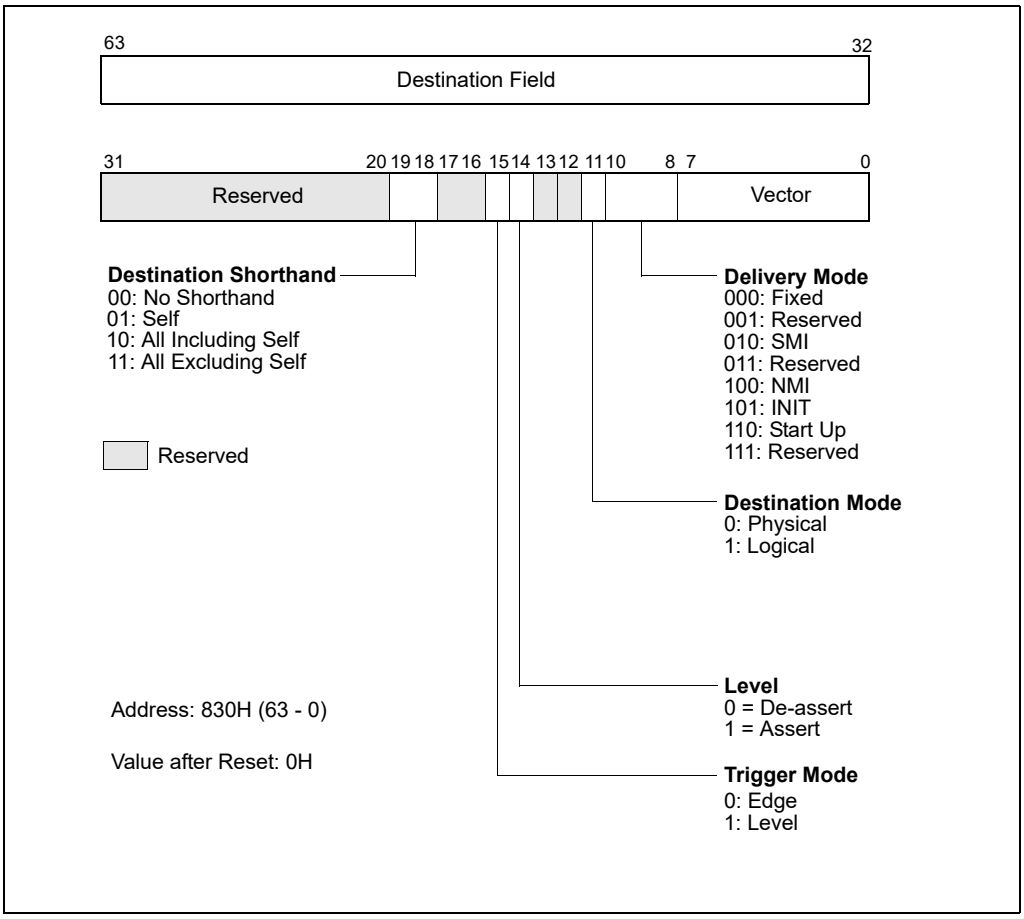


Figure 10-28. Interrupt Command Register (ICR) in x2APIC Mode

A single MSR write to the Interrupt Command Register is required for dispatching an interrupt in x2APIC mode. With the removal of the Delivery Status bit, system software no longer has a reason to read the ICR. It remains readable only to aid in debugging; however, software should not assume the value returned by reading the ICR is the last written value.

A destination ID value of FFFF_FFFFH is used for broadcast of interrupts in both logical destination and physical destination modes.

10.12.10 Determining IPI Destination in x2APIC Mode

10.12.10.1 Logical Destination Mode in x2APIC Mode

In x2APIC mode, the Logical Destination Register (LDR) is increased to 32 bits wide. It is a read-only register to system software. This 32-bit value is referred to as "logical x2APIC ID". System software accesses this register via the RDMSR instruction reading the MSR at address 80DH. Figure 10-29 provides the layout of the Logical Destination Register in x2APIC mode.

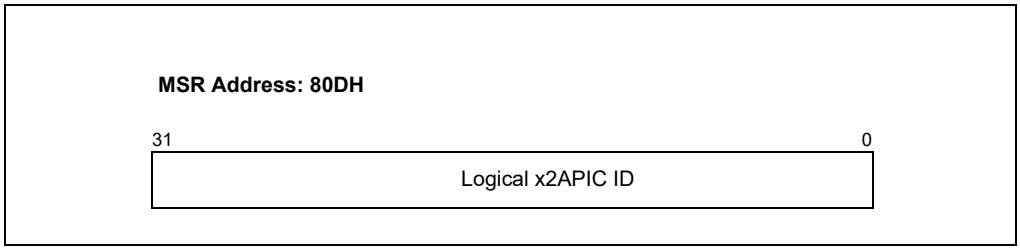


Figure 10-29. Logical Destination Register in x2APIC Mode

In the xAPIC mode, the Destination Format Register (DFR) through the MMIO interface determines the choice of a flat logical mode or a clustered logical mode. Flat logical mode is not supported in the x2APIC mode. Hence the Destination Format Register (DFR) is eliminated in x2APIC mode.

The 32-bit logical x2APIC ID field of LDR is partitioned into two sub-fields:

- Cluster ID (LDR[31:16]): is the address of the destination cluster
- Logical ID (LDR[15:0]): defines a logical ID of the individual local x2APIC within the cluster specified by LDR[31:16].

This layout enables 2^16-1 clusters each with up to 16 unique logical IDs - effectively providing an addressability of ((2^20) - 16) processors in logical destination mode.

It is likely that processor implementations may choose to support less than 16 bits of the cluster ID or less than 16-bits of the Logical ID in the Logical Destination Register. However system software should be agnostic to the number of bits implemented in the cluster ID and logical ID sub-fields. The x2APIC hardware initialization will ensure that the appropriately initialized logical x2APIC IDs are available to system software and reads of non-implemented bits return zero. This is a read-only register that software must read to determine the logical x2APIC ID of the processor. Specifically, software can apply a 16-bit mask to the lowest 16 bits of the logical x2APIC ID to identify the logical address of a processor within a cluster without needing to know the number of implemented bits in cluster ID and Logical ID sub-fields. Similarly, software can create a message destination address for cluster model, by bit-Oring the Logical X2APIC ID (31:0) of processors that have matching Cluster ID(31:16).

To enable cluster ID assignment in a fashion that matches the system topology characteristics and to enable efficient routing of logical mode lowest priority device interrupts in link based platform interconnects, the LDR are initialized by hardware based on the value of x2APIC ID upon x2APIC state transitions. Details of this initialization are provided in Section 10.12.10.2.

10.12.10.2 Deriving Logical x2APIC ID from the Local x2APIC ID

In x2APIC mode, the 32-bit logical x2APIC ID, which can be read from LDR, is derived from the 32-bit local x2APIC ID. Specifically, the 16-bit logical ID sub-field is derived by shifting 1 by the lowest 4 bits of the x2APIC ID, i.e. Logical ID = 1 « x2APIC ID[3:0]. The remaining bits of the x2APIC ID then form the cluster ID portion of the logical x2APIC ID:

```
Logical x2APIC ID = [(x2APIC ID[19:4] « 16) | (1 « x2APIC ID[3:0])]
```

The use of the lowest 4 bits in the x2APIC ID implies that at least 16 APIC IDs are reserved for logical processors within a socket in multi-socket configurations. If more than 16 APIC IDS are reserved for logical processors in a socket/package then multiple cluster IDs can exist within the package.

The LDR initialization occurs whenever the x2APIC mode is enabled (see Section 10.12.5).

10.12.11 SELF IPI Register

SELF IPIs are used extensively by some system software. The x2APIC architecture introduces a new register interface. This new register is dedicated to the purpose of sending self-IPIs with the intent of enabling a highly optimized path for sending self-IPIs.

Figure 10-30 provides the layout of the SELF IPI register. System software only specifies the vector associated with the interrupt to be sent. The semantics of sending a self-IPI via the SELF IPI register are identical to sending a self targeted edge triggered fixed interrupt with the specified vector. Specifically the semantics are identical to the following settings for an inter-processor interrupt sent via the ICR - Destination Shorthand (ICR[19:18] = 01 (Self)), Trigger Mode (ICR[15] = 0 (Edge)), Delivery Mode (ICR[10:8] = 000 (Fixed)), Vector (ICR[7:0] = Vector).

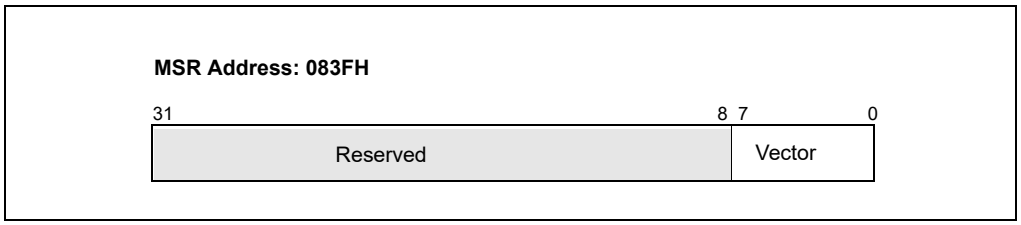


Figure 10-30. SELF IPI register

The SELF IPI register is a write-only register. A RDMSR instruction with address of the SELF IPI register causes a general-protection exception.

The handling and prioritization of a self-IPI sent via the SELF IPI register is architecturally identical to that for an IPI sent via the ICR from a legacy xAPIC unit. Specifically the state of the interrupt would be tracked via the Interrupt Request Register (IRR) and In Service Register (ISR) and Trigger Mode Register (TMR) as if it were received from the system bus. Also sending the IPI via the Self Interrupt Register ensures that interrupt is delivered to the processor core. Specifically completion of the WRMSR instruction to the SELF IPI register implies that the interrupt has been logged into the IRR. As expected for edge triggered interrupts, depending on the processor priority and readiness to accept interrupts, it is possible that interrupts sent via the SELF IPI register or via the ICR with identical vectors can be combined.

10.13 APIC BUS MESSAGE FORMATS

This section describes the message formats used when transmitting messages on the serial APIC bus. The information described here pertains only to the Pentium and P6 family processors.

10.13.1 Bus Message Formats

The local and I/O APICs transmit three types of messages on the serial APIC bus: EOI message, short message, and non-focused lowest priority message. The purpose of each type of message and its format are described below.

10.13.2 EOI Message

Local APICs send 14-cycle EOI messages to the I/O APIC to indicate that a level triggered interrupt has been accepted by the processor. This interrupt, in turn, is a result of software writing into the EOI register of the local APIC. Table 10-8 shows the cycles in an EOI message.

Table 10-8. EOI Message (14 Cycles)

Cycle	Bit1	Bit0	
1	1	1	11 = EOI
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	

Table 10-8. EOI Message (14 Cycles) (Contd.)

Cycle	Bit1	Bit0	
4	ArbID1	0	
5	ArbID0	0	
6	V7	V6	Interrupt vector V7 - V0
7	V5	V4	
8	V3	V2	
9	V1	V0	
10	C	C	Checksum for cycles 6 - 9
11	0	0	
12	A	A	Status Cycle 0
13	A1	A1	Status Cycle 1
14	0	0	Idle

The checksum is computed for cycles 6 through 9. It is a cumulative sum of the 2-bit (Bit1:Bit0) logical data values. The carry out of all but the last addition is added to the sum. If any APIC computes a different checksum than the one appearing on the bus in cycle 10, it signals an error, driving 11 on the APIC bus during cycle 12. In this case, the APICs disregard the message. The sending APIC will receive an appropriate error indication (see Section 10.5.3, "Error Handling") and resend the message. The status cycles are defined in Table 10-11.

10.13.2.1 Short Message

Short messages (21-cycles) are used for sending fixed, NMI, SMI, INIT, start-up, ExtINT and lowest-priority-with-focus interrupts. Table 10-9 shows the cycles in a short message.

Table 10-9. Short Message (21 Cycles)

Cycle	Bit1	Bit0	
1	0	1	0 1 = normal
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = Destination Mode
7	M1	M0	M2-M0 = Delivery mode
8	L	TM	L = Level, TM = Trigger Mode
9	V7	V6	V7-V0 = Interrupt Vector
10	V5	V4	
11	V3	V2	
12	V1	V0	
13	D7	D6	D7-D0 = Destination
14	D5	D4	
15	D3	D2	
16	D1	D0	
17	C	C	Checksum for cycles 6-16

Table 10-9. Short Message (21 Cycles) (Contd.)

Cycle	Bit1	Bit0	
18	0	0	
19	A	A	Status cycle 0
20	A1	A1	Status cycle 1
21	0	0	Idle

If the physical delivery mode is being used, then cycles 15 and 16 represent the APIC ID and cycles 13 and 14 are considered don't care by the receiver. If the logical delivery mode is being used, then cycles 13 through 16 are the 8-bit logical destination field.

For shorthands of "all-incl-self" and "all-excl-self," the physical delivery mode and an arbitration priority of 15 (D0:D3 = 1111) are used. The agent sending the message is the only one required to distinguish between the two cases. It does so using internal information.

When using lowest priority delivery with an existing focus processor, the focus processor identifies itself by driving 10 during cycle 19 and accepts the interrupt. This is an indication to other APICs to terminate arbitration. If the focus processor has not been found, the short message is extended on-the-fly to the non-focused lowest-priority message. Note that except for the EOI message, messages generating a checksum or an acceptance error (see Section 10.5.3, "Error Handling") terminate after cycle 21.

10.13.2.2 Non-focused Lowest Priority Message

These 34-cycle messages (see Table 10-10) are used in the lowest priority delivery mode when a focus processor is not present. Cycles 1 through 20 are same as for the short message. If during the status cycle (cycle 19) the state of the (A:A) flags is 10B, a focus processor has been identified, and the short message format is used (see Table 10-9). If the (A:A) flags are set to 00B, lowest priority arbitration is started and the 34-cycles of the non-focused lowest priority message are competed. For other combinations of status flags, refer to Section 10.13.2.3, "APIC Bus Status Cycles."

Table 10-10. Non-Focused Lowest Priority Message (34 Cycles)

Cycle	Bit0	Bit1	
1	0	1	0 1 = normal
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = Destination mode
7	M1	M0	M2-M0 = Delivery mode
8	L	TM	L = Level, TM = Trigger Mode
9	V7	V6	V7-V0 = Interrupt Vector
10	V5	V4	
11	V3	V2	
12	V1	V0	
13	D7	D6	D7-D0 = Destination
14	D5	D4	
15	D3	D2	
16	D1	D0	
17	C	C	Checksum for cycles 6-16

Table 10-10. Non-Focused Lowest Priority Message (34 Cycles) (Contd.)

Cycle	Bit0	Bit1	
18	0	0	
19	A	A	Status cycle 0
20	A1	A1	Status cycle 1
21	P7	0	P7 - P0 = Inverted Processor Priority
22	P6	0	
23	P5	0	
24	P4	0	
25	P3	0	
26	P2	0	
27	P1	0	
28	P0	0	
29	ArbID3	0	Arbitration ID 3 -0
30	ArbID2	0	
31	ArbID1	0	
32	ArbID0	0	
33	A2	A2	Status Cycle
34	0	0	Idle

Cycles 21 through 28 are used to arbitrate for the lowest priority processor. The processors participating in the arbitration drive their inverted processor priority on the bus. Only the local APICs having free interrupt slots participate in the lowest priority arbitration. If no such APIC exists, the message will be rejected, requiring it to be tried at a later time.

Cycles 29 through 32 are also used for arbitration in case two or more processors have the same lowest priority. In the lowest priority delivery mode, all combinations of errors in cycle 33 (A2 A2) will set the "accept error" bit in the error status register (see Figure 10-9). Arbitration priority update is performed in cycle 20, and is not affected by errors detected in cycle 33. Only the local APIC that wins in the lowest priority arbitration, drives cycle 33. An error in cycle 33 will force the sender to resend the message.

10.13.2.3 APIC Bus Status Cycles

Certain cycles within an APIC bus message are status cycles. During these cycles the status flags (A:A) and (A1:A1) are examined. Table 10-11 shows how these status flags are interpreted, depending on the current delivery mode and existence of a focus processor.

Table 10-11. APIC Bus Status Cycles Interpretation

Delivery Mode	A Status	A1 Status	A2 Status	Update ArbID and Cycle#	Message Length	Retry
EOI	00: CS_OK	10: Accept	XX:	Yes, 13	14 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 13	14 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	14 Cycle	Yes
	11: CS_Error	XX:	XX:	No	14 Cycle	Yes
	10: Error	XX:	XX:	No	14 Cycle	Yes
	01: Error	XX:	XX:	No	14 Cycle	Yes
Fixed	00: CS_OK	10: Accept	XX:	Yes, 20	21 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 Cycle	Yes
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	10: Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes
NMI, SMI, INIT, ExtINT, Start-Up	00: CS_OK	10: Accept	XX:	Yes, 20	21 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 Cycle	Yes
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	10: Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes
Lowest	00: CS_OK, NoFocus	11: Do Lowest	10: Accept	Yes, 20	34 Cycle	No
	00: CS_OK, NoFocus	11: Do Lowest	11: Error	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	11: Do Lowest	0X: Error	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	10: End and Retry	XX:	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	0X: Error	XX:	No	34 Cycle	Yes
	10: CS_OK, Focus	XX:	XX:	Yes, 20	34 Cycle	No
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes

15. Updates to Chapter 14, Volume 3B

Change bars and green text show changes to Chapter 14 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

Changes to this chapter: Added details on Intel® Thread Director.

CHAPTER 14
POWER AND THERMAL MANAGEMENT

This chapter describes facilities of Intel 64 and IA-32 architecture used for power management and thermal monitoring.

14.1 ENHANCED INTEL SPEEDSTEP® TECHNOLOGY

Enhanced Intel SpeedStep® Technology was introduced in the Pentium M processor. The technology enables the management of processor power consumption via performance state transitions. These states are defined as discrete operating points associated with different voltages and frequencies.

Enhanced Intel SpeedStep Technology differs from previous generations of Intel SpeedStep® Technology in two ways:

- Centralization of the control mechanism and software interface in the processor by using model-specific registers.
- Reduced hardware overhead; this permits more frequent performance state transitions.

Previous generations of the Intel SpeedStep Technology require processors to be a deep sleep state, holding off bus master transfers for the duration of a performance state transition. Performance state transitions under the Enhanced Intel SpeedStep Technology are discrete transitions to a new target frequency.

Support is indicated by CPUID, using ECX feature bit 07. Enhanced Intel SpeedStep Technology is enabled by setting IA32_MISC_ENABLE MSR, bit 16. On reset, bit 16 of IA32_MISC_ENABLE MSR is cleared.

14.1.1 Software Interface For Initiating Performance State Transitions

State transitions are initiated by writing a 16-bit value to the IA32_PERF_CTL register, see Figure 14-2. If a transition is already in progress, transition to a new value will subsequently take effect.

Reads of IA32_PERF_CTL determine the last targeted operating point. The current operating point can be read from IA32_PERF_STATUS. IA32_PERF_STATUS is updated dynamically.

The 16-bit encoding that defines valid operating points is model-specific. Applications and performance tools are not expected to use either IA32_PERF_CTL or IA32_PERF_STATUS and should treat both as reserved. Performance monitoring tools can access model-specific events and report the occurrences of state transitions.

14.2 P-STATE HARDWARE COORDINATION

The Advanced Configuration and Power Interface (ACPI) defines performance states (P-states) that are used to facilitate system software's ability to manage processor power consumption. Different P-states correspond to different performance levels that are applied while the processor is actively executing instructions. Enhanced Intel SpeedStep Technology supports P-states by providing software interfaces that control the operating frequency and voltage of a processor.

With multiple processor cores residing in the same physical package, hardware dependencies may exist for a subset of logical processors on a platform. These dependencies may impose requirements that impact the coordination of P-state transitions. As a result, multi-core processors may require an OS to provide additional software support for coordinating P-state transitions for those subsets of logical processors.

ACPI firmware can choose to expose P-states as dependent and hardware-coordinated to OS power management (OSPM) policy. To support OSPMs, multi-core processors must have additional built-in support for P-state hardware coordination and feedback.

Intel 64 and IA-32 processors with dependent P-states amongst a subset of logical processors permit hardware coordination of P-states and provide a hardware-coordination feedback mechanism using IA32_MPERF MSR and

IA32_APERF MSR. See Figure 14-1 for an overview of the two 64-bit MSRs and the bullets below for a detailed description.

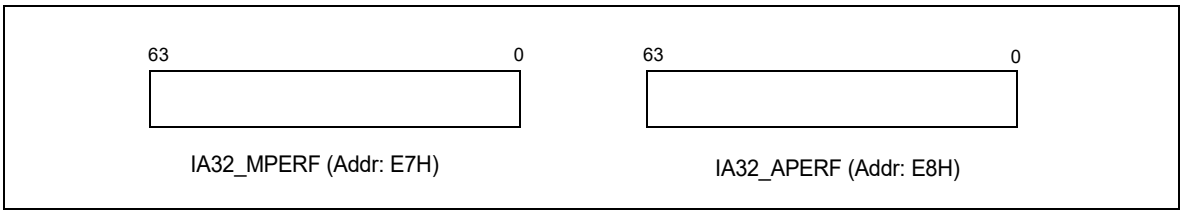


Figure 14-1. IA32_MPERF MSR and IA32_APERF MSR for P-state Coordination

- Use CPUID to check the P-State hardware coordination feedback capability bit. CPUID.06H.ECX[Bit 0] = 1 indicates IA32_MPERF MSR and IA32_APERF MSR are present.
- IA32_MPERF MSR (E7H) increments in proportion to a fixed frequency, which is configured when the processor is booted.
- IA32_APERF MSR (E8H) increments in proportion to actual performance, while accounting for hardware coordination of P-state and TM1/TM2; or software initiated throttling.
- The MSRs are per logical processor; they measure performance only when the targeted processor is in the C0 state.
- Only the IA32_APERF/IA32_MPERF ratio is architecturally defined; software should not attach meaning to the content of the individual of IA32_APERF or IA32_MPERF MSRs.
- When either MSR overflows, both MSRs are reset to zero and continue to increment.
- Both MSRs are full 64-bits counters. Each MSR can be written to independently. However, software should follow the guidelines illustrated in Example 14-1.

If P-states are exposed by the BIOS as hardware coordinated, software is expected to confirm processor support for P-state hardware coordination feedback and use the feedback mechanism to make P-state decisions. The OSPM is expected to either save away the current MSR values (for determination of the delta of the counter ratio at a later time) or reset both MSRs (execute WRMSR with 0 to these MSRs individually) at the start of the time window used for making the P-state decision. When not resetting the values, overflow of the MSRs can be detected by checking whether the new values read are less than the previously saved values.

Example 14-1 demonstrates steps for using the hardware feedback mechanism provided by IA32_APERF MSR and IA32_MPERF MSR to determine a target P-state.

Example 14-1. Determine Target P-state From Hardware Coordinated Feedback

```
DWORD PercentBusy; // Percentage of processor time not idle.
    // Measure "PercentBusy" during previous sampling window.
    // Typically, "PercentBusy" is measure over a time scale suitable for
    // power management decisions
    //
    // RDMSR of MCNT and ACNT should be performed without delay.
    // Software needs to exercise care to avoid delays between
    // the two RDMSRs (for example, interrupts).
    MCNT = RDMSR(IA32_MPERF);
    ACNT = RDMSR(IA32_APERF);

    // PercentPerformance indicates the percentage of the processor
    // that is in use. The calculation is based on the PercentBusy,
    // that is the percentage of processor time not idle and the P-state
    // hardware coordinated feedback using the ACNT/MCNT ratio.
    // Note that both values need to be calculated over the same
```

```
// time window.
      PercentPerformance = PercentBusy * (ACNT/MCNT);


// This example does not cover the additional logic or algorithms
//  necessary to coordinate multiple logical processors to a target P-state.

TargetPstate = FindPstate(PercentPerformance);

if (TargetPstate ≠ currentPstate) {
      SetPState(TargetPstate);
}
// WRMSR of MCNT and ACNT should be performed without delay.
// Software needs to exercise care to avoid delays between
// the two WRMSRs (for example, interrupts).
 WRMSR(IA32_MPERF, 0);
 WRMSR(IA32_APERF, 0);
```

14.3 SYSTEM SOFTWARE CONSIDERATIONS AND OPPORTUNISTIC PROCESSOR PERFORMANCE OPERATION

An Intel 64 processor may support a form of processor operation that takes advantage of design headroom to opportunistically increase performance. The Intel® Turbo Boost Technology can convert thermal headroom into higher performance across multi-threaded and single-threaded workloads. The Intel® Dynamic Acceleration Technology feature can convert thermal headroom into higher performance if only one thread is active.

14.3.1 Intel® Dynamic Acceleration Technology

The Intel Core 2 Duo processor T 7700 introduces Intel Dynamic Acceleration Technology. Intel Dynamic Acceleration Technology takes advantage of thermal design headroom and opportunistically allows a single core to operate at a higher performance level when the operating system requests increased performance.

14.3.2 System Software Interfaces for Opportunistic Processor Performance Operation

Opportunistic processor performance operation, applicable to Intel Dynamic Acceleration Technology and Intel® Turbo Boost Technology, has the following characteristics:

- A transition from a normal state of operation (e.g. Intel Dynamic Acceleration Technology/Turbo mode disengaged) to a target state is not guaranteed, but may occur opportunistically after the corresponding enable mechanism is activated, the headroom is available and certain criteria are met.
- The opportunistic processor performance operation is generally transparent to most application software.
- System software (BIOS and Operating system) must be aware of hardware support for opportunistic processor performance operation and may need to temporarily disengage opportunistic processor performance operation when it requires more predictable processor operation.
- When opportunistic processor performance operation is engaged, the OS should use hardware coordination feedback mechanisms to prevent un-intended policy effects if it is activated during inappropriate situations.

14.3.2.1 Discover Hardware Support and Enabling of Opportunistic Processor Performance Operation

If an Intel 64 processor has hardware support for opportunistic processor performance operation, the power-on default state of IA32_MISC_ENABLE[38] indicates the presence of such hardware support. For Intel 64 processors that support opportunistic processor performance operation, the default value is 1, indicating its presence. For processors that do not support opportunistic processor performance operation, the default value is 0. The power-

on default value of IA32_MISC_ENABLE[38] allows BIOS to detect the presence of hardware support of opportunistic processor performance operation.

IA32_MISC_ENABLE[38] is shared across all logical processors in a physical package. It is written by BIOS during platform initiation to enable/disable opportunistic processor performance operation in conjunction of OS power management capabilities, see Section 14.3.2.2. BIOS can set IA32_MISC_ENABLE[38] with 1 to disable opportunistic processor performance operation; it must clear the default value of IA32_MISC_ENABLE[38] to 0 to enable opportunistic processor performance operation. OS and applications must use CPUID leaf 06H if it needs to detect processors that have opportunistic processor performance operation enabled.

When CPUID is executed with EAX = 06H on input, Bit 1 of EAX in Leaf 06H (i.e. CPUID.06H:EAX[1]) indicates opportunistic processor performance operation, such as Intel Dynamic Acceleration Technology, has been enabled by BIOS.

Opportunistic processor performance operation can be disabled by setting bit 38 of IA32_MISC_ENABLE. This mechanism is intended for BIOS only. If IA32_MISC_ENABLE[38] is set, CPUID.06H:EAX[1] will return 0.

14.3.2.2 OS Control of Opportunistic Processor Performance Operation

There may be phases of software execution in which system software cannot tolerate the non-deterministic aspects of opportunistic processor performance operation. For example, when calibrating a real-time workload to make a CPU reservation request to the OS, it may be undesirable to allow the possibility of the processor delivering increased performance that cannot be sustained after the calibration phase.

System software can temporarily disengage opportunistic processor performance operation by setting bit 32 of the IA32_PERF_CTL MSR (0199H), using a read-modify-write sequence on the MSR. The opportunistic processor performance operation can be re-engaged by clearing bit 32 in IA32_PERF_CTL MSR, using a read-modify-write sequence. The DISENAGE bit in IA32_PERF_CTL is not reflected in bit 32 of the IA32_PERF_STATUS MSR (0198H), and it is not shared between logical processors in a physical package. In order for OS to engage Intel Dynamic Acceleration Technology/Turbo mode, the BIOS must:

- Enable opportunistic processor performance operation, as described in Section 14.3.2.1.
- Expose the operating points associated with Intel Dynamic Acceleration Technology/Turbo mode to the OS.

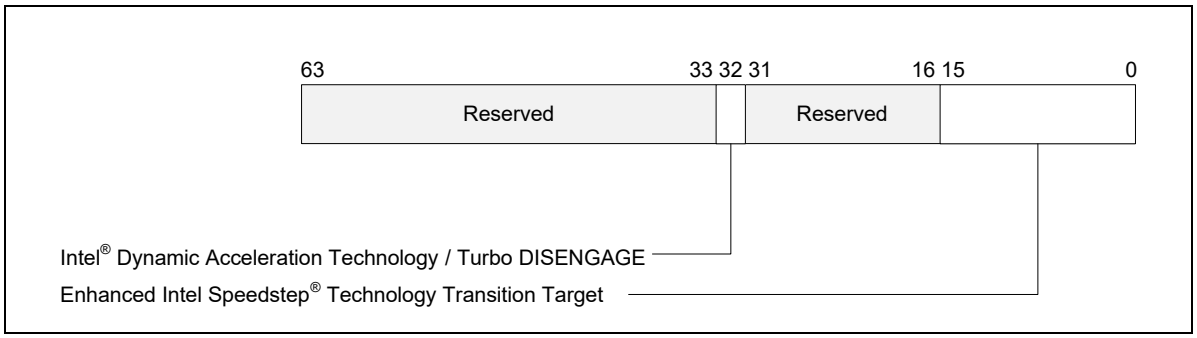


Figure 14-2. IA32_PERF_CTL Register

14.3.2.3 Required Changes to OS Power Management P-State Policy

Intel Dynamic Acceleration Technology and Intel Turbo Boost Technology can provide opportunistic performance greater than the performance level corresponding to the Processor Base frequency of the processor (see CPUID's processor frequency information). System software can use a pair of MSRs to observe performance feedback. Software must query for the presence of IA32_APERF and IA32_MPERF (see Section 14.2). The ratio between IA32_APERF and IA32_MPERF is architecturally defined and a value greater than unity indicates performance increase occurred during the observation period due to Intel Dynamic Acceleration Technology. Without incorporating such performance feedback, the target P-state evaluation algorithm can result in a non-optimal P-state target.

There are other scenarios under which OS power management may want to disable Intel Dynamic Acceleration Technology, some of these are listed below:

- When engaging ACPI defined passive thermal management, it may be more effective to disable Intel Dynamic Acceleration Technology for the duration of passive thermal management.
- When the user has indicated a policy preference of power savings over performance, OS power management may want to disable Intel Dynamic Acceleration Technology while that policy is in effect.

14.3.3 Intel® Turbo Boost Technology

Intel Turbo Boost Technology is supported in Intel Core i7 processors and Intel Xeon processors based on Nehalem microarchitecture. It uses the same principle of leveraging thermal headroom to dynamically increase processor performance for single-threaded and multi-threaded/multi-tasking environment. The programming interface described in Section 14.3.2 also applies to Intel Turbo Boost Technology.

14.3.4 Performance and Energy Bias Hint Support

Intel 64 processors may support additional software hint to guide the hardware heuristic of power management features to favor increasing dynamic performance or conserve energy consumption.

Software can detect the processor's capability to support the performance-energy bias preference hint by examining bit 3 of ECX in CPUID leaf 6. The processor supports this capability if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H).

Software can program the lowest four bits of IA32_ENERGY_PERF_BIAS MSR with a value from 0 - 15. The values represent a sliding scale, where a value of 0 (the default reset value) corresponds to a hint preference for highest performance and a value of 15 corresponds to the maximum energy savings. A value of 7 roughly translates into a hint to balance performance with energy consumption.

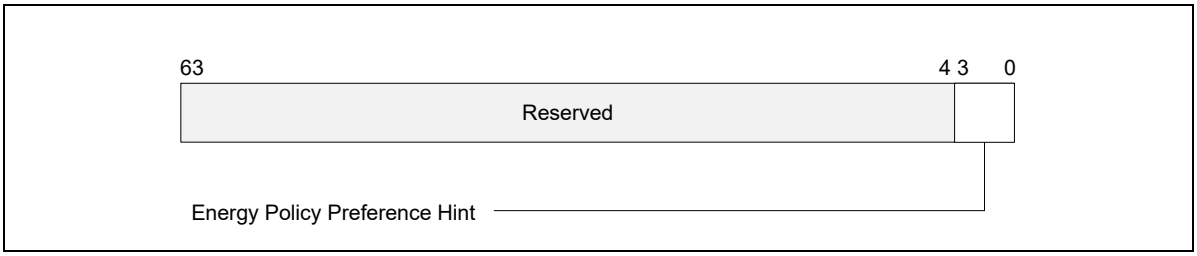


Figure 14-3. IA32_ENERGY_PERF_BIAS Register

The layout of IA32_ENERGY_PERF_BIAS is shown in Figure 14-3. The scope of IA32_ENERGY_PERF_BIAS is per logical processor, which means that each of the logical processors in the package can be programmed with a different value. This may be especially important in virtualization scenarios, where the performance / energy requirements of one logical processor may differ from the other. Conflicting "hints" from various logical processors at higher hierarchy level will be resolved in favor of performance over energy savings.

Software can use whatever criteria it sees fit to program the MSR with an appropriate value. However, the value only serves as a hint to the hardware and the actual impact on performance and energy savings is model specific.

14.4 HARDWARE-CONTROLLED PERFORMANCE STATES (HWP)

Intel processors may contain support for Hardware-Controlled Performance States (HWP), which autonomously selects performance states while utilizing OS supplied performance guidance hints. The Enhanced Intel Speed-Step® Technology provides a means for the OS to control and monitor discrete frequency-based operating points via the IA32_PERF_CTL and IA32_PERF_STATUS MSRs.

In contrast, HWP is an implementation of the ACPI-defined Collaborative Processor Performance Control (CPPC), which specifies that the platform enumerates a continuous, abstract unit-less, performance value scale that is not tied to a specific performance state / frequency by definition. While the enumerated scale is roughly linear in terms of a delivered integer workload performance result, the OS is required to characterize the performance value range to comprehend the delivered performance for an applied workload.

When HWP is enabled, the processor autonomously selects performance states as deemed appropriate for the applied workload and with consideration of constraining hints that are programmed by the OS. These OS-provided hints include minimum and maximum performance limits, preference towards energy efficiency or performance, and the specification of a relevant workload history observation time window. The means for the OS to override HWP's autonomous selection of performance state with a specific desired performance target is also provided, however, the effective frequency delivered is subject to the result of energy efficiency and performance optimizations.

14.4.1 HWP Programming Interfaces

The programming interfaces provided by HWP include the following:

- The CPUID instruction allows software to discover the presence of HWP support in an Intel processor. Specifically, execute CPUID instruction with EAX=06H as input will return 5 bit flags covering the following aspects in bits 7 through 11 of CPUID.06H:EAX:
  - Availability of HWP baseline resource and capability, CPUID.06H:EAX[bit 7]: If this bit is set, HWP provides several new architectural MSRs: IA32_PM_ENABLE, IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS.
  - Availability of HWP Notification upon dynamic Guaranteed Performance change, CPUID.06H:EAX[bit 8]: If this bit is set, HWP provides IA32_HWP_INTERRUPT MSR to enable interrupt generation due to dynamic Performance changes and excursions.
  - Availability of HWP Activity window control, CPUID.06H:EAX[bit 9]: If this bit is set, HWP allows software to program activity window in the IA32_HWP_REQUEST MSR.
  - Availability of HWP energy/performance preference control, CPUID.06H:EAX[bit 10]: If this bit is set, HWP allows software to set an energy/performance preference hint in the IA32_HWP_REQUEST MSR.
  - Availability of HWP package level control, CPUID.06H:EAX[bit 11]:If this bit is set, HWP provides the IA32_HWP_REQUEST_PKG MSR to convey OS Power Management's control hints for all logical processors in the physical package.

Table 14-1. Architectural and Non-Architectural MSRs Related to HWP

Address	Architectural	Register Name	Description
770H	Y	IA32_PM_ENABLE	Enable/Disable HWP.
771H	Y	IA32_HWP_CAPABILITIES	Enumerates the HWP performance range (static and dynamic).
772H	Y	IA32_HWP_REQUEST_PKG	Conveys OSPM's control hints (Min, Max, Activity Window, Energy Performance Preference, Desired) for all logical processor in the physical package.
773H	Y	IA32_HWP_INTERRUPT	Controls HWP native interrupt generation (Guaranteed Performance changes, excursions).
774H	Y	IA32_HWP_REQUEST	Conveys OSPM's control hints (Min, Max, Activity Window, Energy Performance Preference, Desired) for a single logical processor.
775H	Y	IA32_HWP_PECI_REQUEST_INFO	Conveys embedded system controller requests to override some of the OS HWP Request settings via the PECI mechanism.
777H	Y	IA32_HWP_STATUS	Status bits indicating changes to Guaranteed Performance and excursions to Minimum Performance.
19CH	Y	IA32_THERM_STATUS[bits 15:12]	Conveys reasons for performance excursions.
64EH	N	MSR_PPERF	Productive Performance Count.

- Additionally, HWP may provide a non-architectural MSR, MSR_PPERF, which provides a quantitative metric to software of hardware's view of workload scalability. This hardware's view of workload scalability is implementation specific.

14.4.2 Enabling HWP

The layout of the IA32_PM_ENABLE MSR is shown in Figure 14-4. The bit fields are described below:

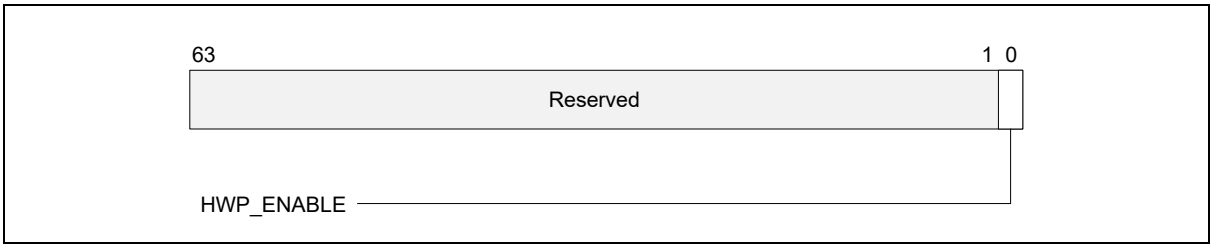


Figure 14-4. IA32_PM_ENABLE MSR

- **HWP_ENABLE (bit 0, R/W1Once)** — Software sets this bit to enable HWP with autonomous selection of processor P-States. When set, the processor will disregard input from the legacy performance control interface (IA32_PERF_CTL). 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 = zero (0).
- Bits 63:1 are reserved and must be zero.

After software queries CPUID and verifies the processor's support of HWP, system software can write 1 to IA32_PM_ENABLE.HWP_ENABLE (bit 0) to enable hardware controlled performance states. The default value of IA32_PM_ENABLE MSR at power-on is 0, i.e. HWP is disabled.

Additional MSRs associated with HWP may only be accessed after HWP is enabled, with the exception of IA32_HWP_INTERRUPT and MSR_PPERF. Accessing the IA32_HWP_INTERRUPT MSR requires only HWP is present as enumerated by CPUID but does not require enabling HWP.

IA32_PM_ENABLE is a package level MSR, i.e., writing to it from any logical processor within a package affects all logical processors within that package.

14.4.3 HWP Performance Range and Dynamic Capabilities

The OS reads the IA32_HWP_CAPABILITIES MSR to comprehend the limits of the HWP-managed performance range as well as the dynamic capability, which may change during processor operation. The enumerated performance range values reported by IA32_HWP_CAPABILITIES directly map to initial frequency targets (prior to workload-specific frequency optimizations of HWP). However the mapping is processor family specific.

The layout of the IA32_HWP_CAPABILITIES MSR is shown in Figure 14-5. The bit fields are described below:

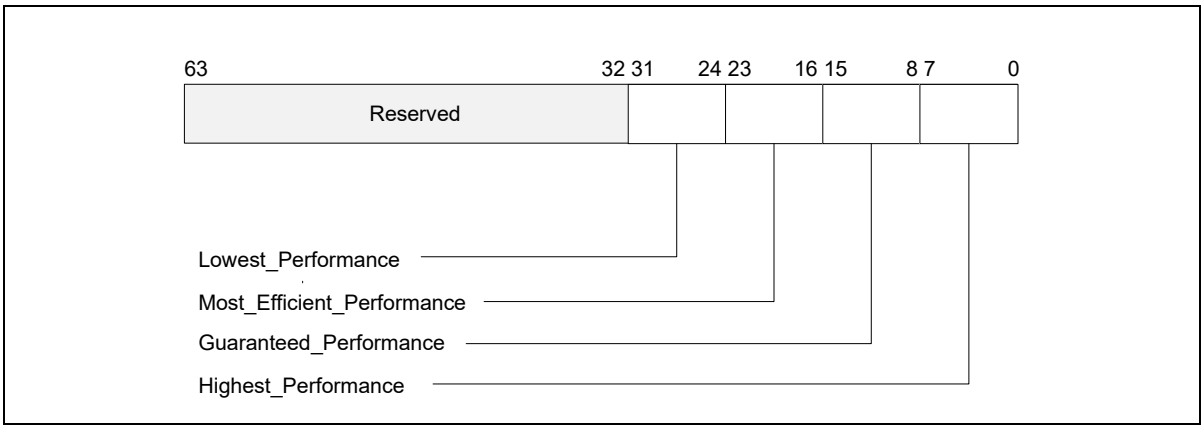


Figure 14-5. IA32_HWP_CAPABILITIES Register

- **Highest_Performance (bits 7:0, RO)** — Value for the maximum non-guaranteed performance level.
- **Guaranteed_Performance (bits 15:8, RO)** — Current value for the guaranteed performance level. This value can change dynamically as a result of internal or external constraints, e.g. thermal or power limits.
- **Most_Efficient_Performance (bits 23:16, RO)** — Current value of the most efficient performance level. This value can change dynamically as a result of workload characteristics.
- **Lowest_Performance (bits 31:24, RO)** — Value for the lowest performance level that software can program to IA32_HWP_REQUEST.
- Bits 63:32 are reserved and must be zero.

The value returned in the **Guaranteed_Performance** field is hardware's best-effort approximation of the available performance given current operating constraints. Changes to the Guaranteed_Performance value will primarily occur due to a shift in operational mode. This includes a power or other limit applied by an external agent, e.g. RAPL (see Figure 14.10.1), or the setting of a Configurable TDP level (see model-specific controls related to Programmable TDP Limit in Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*.). Notification of a change to the Guaranteed_Performance occurs via interrupt (if configured) and the IA32_HWP_Status MSR. Changes to Guaranteed_Performance are indicated when a macroscopically meaningful change in performance occurs i.e. sustained for greater than one second. Consequently, notification of a change in Guaranteed Performance will typically occur no more frequently than once per second. Rapid changes in platform configuration, e.g. docking / undocking, with corresponding changes to a Configurable TDP level could potentially cause more frequent notifications.

The value returned by the **Most_Efficient_Performance** field provides the OS with an indication of the practical lower limit for the IA32_HWP_REQUEST. The processor may not honor IA32_HWP_REQUEST.Maximum Performance settings below this value.

14.4.4 Managing HWP

14.4.4.1 IA32_HWP_REQUEST MSR (Address: 774H Logical Processor Scope)

Typically, the operating system controls HWP operation for each logical processor via the writing of control hints / constraints to the IA32_HWP_REQUEST MSR. The layout of the IA32_HWP_REQUEST MSR is shown in Figure 14-6. The bit fields are described below Figure 14-6.

Operating systems can control HWP by writing both IA32_HWP_REQUEST and IA32_HWP_REQUEST_PKG MSRs (see Section 14.4.4.2). Five valid bits within the IA32_HWP_REQUEST MSR let the operating system flexibly select which of its five hint / constraint fields should be derived by the processor from the IA32_HWP_REQUEST MSR and which should be derived from the IA32_HWP_REQUEST_PKG MSR. These five valid bits are supported if CPUID[6].EAX[17] is set.

When the IA32_HWP_REQUEST MSR Package Control bit is set, any valid bit that is NOT set indicates to the processor to use the respective field value from the IA32_HWP_REQUEST_PKG MSR. Otherwise, the values are derived from the IA32_HWP_REQUEST MSR. The valid bits are ignored when the IA32_HWP_REQUEST MSR Package Control bit is zero.

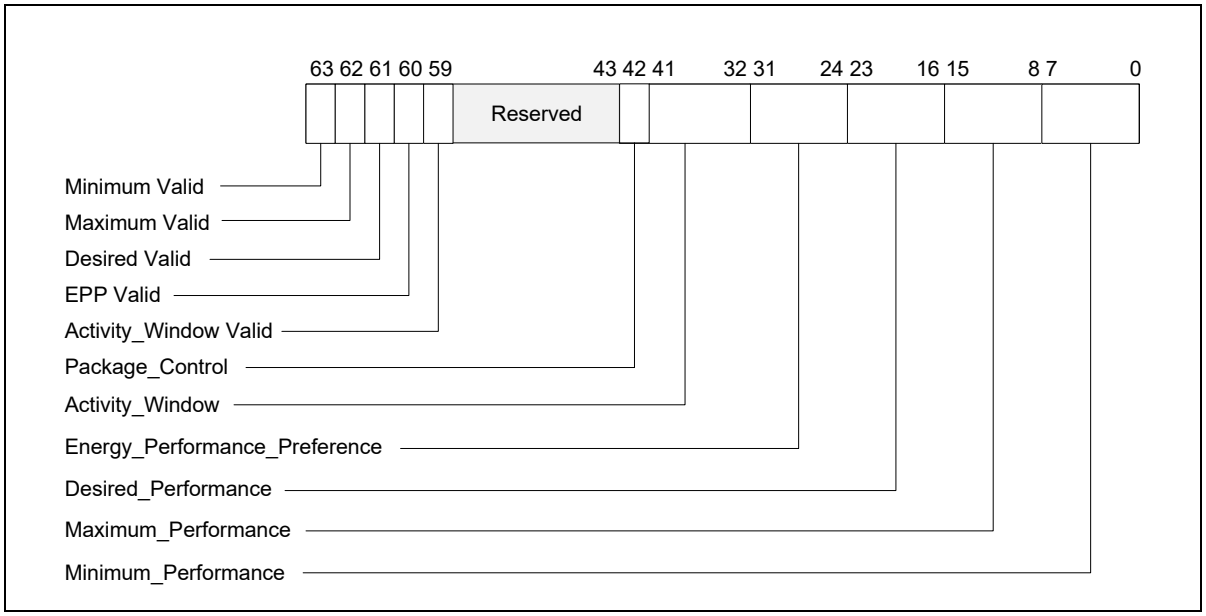


Figure 14-6. IA32_HWP_REQUEST Register

- **Minimum_Performance (bits 7:0, RW)** — Conveys a hint to the HWP hardware. The OS programs the minimum performance hint to achieve the required quality of service (QOS) or to meet a service level agreement (SLA) as needed. Note that an excursion below the level specified is possible due to hardware constraints. The default value of this field is IA32_HWP_CAPABILITIES.Lowest_Performance.
- **Maximum_Performance (bits 15:8, RW)** — Conveys a hint to the HWP hardware. The OS programs this field to limit the maximum performance that is expected to be supplied by the HWP hardware. Excursions above the limit requested by OS are possible due to hardware coordination between the processor cores and other components in the package. The default value of this field is IA32_HWP_CAPABILITIES.Highest_Performance.
- **Desired_Performance (bits 23:16, RW)** — Conveys a hint to the HWP hardware. When set to zero, hardware autonomous selection determines the performance target. When set to a non-zero value (between the range of Lowest_Performance and Highest_Performance of IA32_HWP_CAPABILITIES) conveys an explicit performance request hint to the hardware; effectively disabling HW Autonomous selection. The Desired_Performance input is non-constraining in terms of Performance and Energy Efficiency optimizations, which are independently controlled. The default value of this field is 0.
- **Energy_Performance_Preference (bits 31:24, RW)** — Conveys a hint to the HWP hardware. The OS may write a range of values from 0 (performance preference) to 0FFH (energy efficiency preference) to influence the rate of performance increase /decrease and the result of the hardware's energy efficiency and performance optimizations. The default value of this field is 80H. Note: If CPUID.06H:EAX[bit 10] indicates that this field is not supported, HWP uses the value of the IA32_ENERGY_PERF_BIAS MSR to determine the energy efficiency / performance preference.
- **Activity_Window (bits 41:32, RW)** — Conveys a hint to the HWP hardware specifying a moving workload history observation window for performance/frequency optimizations. If 0, the hardware will determine the appropriate window size. When writing a non-zero value to this field, this field is encoded in the format of bits 38:32 as a 7-bit mantissa and bits 41:39 as a 3-bit exponent value in powers of 10. The resultant value is in microseconds. Thus, the minimal/maximum activity window size is 1 microsecond/1270 seconds. Combined with the Energy_Performance_Preference input, Activity_Window influences the rate of performance increase

/ decrease. This non-zero hint only has meaning when Desired_Performance = 0. The default value of this field is 0.

- **Package_Control (bit 42, RW)** — When set, causes this logical processor's IA32_HWP_REQUEST control inputs to be derived from the IA32_HWP_REQUEST_PKG MSR.
- Bits 58:43 are reserved and must be zero.
- **Activity_Window Valid (bit 59, RW)** — When set, indicates to the processor to derive the Activity Window field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **EPP Valid (bit 60, RW)** — When set, indicates to the processor to derive the EPP field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Desired Valid (bit 61, RW)** — When set, indicates to the processor to derive the Desired Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Maximum Valid (bit 62, RW)** — When set, indicates to the processor to derive the Maximum Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Minimum Valid (bit 63, RW)** — When set, indicates to the processor to derive the Minimum Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.

The HWP hardware clips and resolves the field values as necessary to the valid range. Reads return the last value written not the clipped values.

Processors may support a subset of IA32_HWP_REQUEST fields as indicated by CPUID. Reads of non-supported fields will return 0. Writes to non-supported fields are ignored.

The OS may override HWP's autonomous selection of performance state with a specific performance target by setting the Desired_Performance field to a non-zero value, however, the effective frequency delivered is subject to the result of energy efficiency and performance optimizations, which are influenced by the Energy Performance Preference field.

Software may disable all hardware optimizations by setting Minimum_Performance = Maximum_Performance (subject to package coordination).

Note: The processor may run below the Minimum_Performance level due to hardware constraints including: power, thermal, and package coordination constraints. The processor may also run below the Minimum_Performance level for short durations (few milliseconds) following C-state exit, and when Hardware Duty Cycling (see Section 14.5) is enabled.

When the IA32_HWP_REQUEST MSR is set to fast access mode, writes of this MSR are posted, i.e., the WRMSR instruction retires before the data reaches its destination within the processor. It may retire even before all preceding IA stores are globally visible, i.e., it is not an architecturally serializing instruction anymore (no store fence). A new CPUID bit indicates this new characteristic of the IA32_HWP_REQUEST MSR (see Section 14.4.8 for additional details).

14.4.4.2 IA32_HWP_REQUEST_PKG MSR (Address: 772H Package Scope)

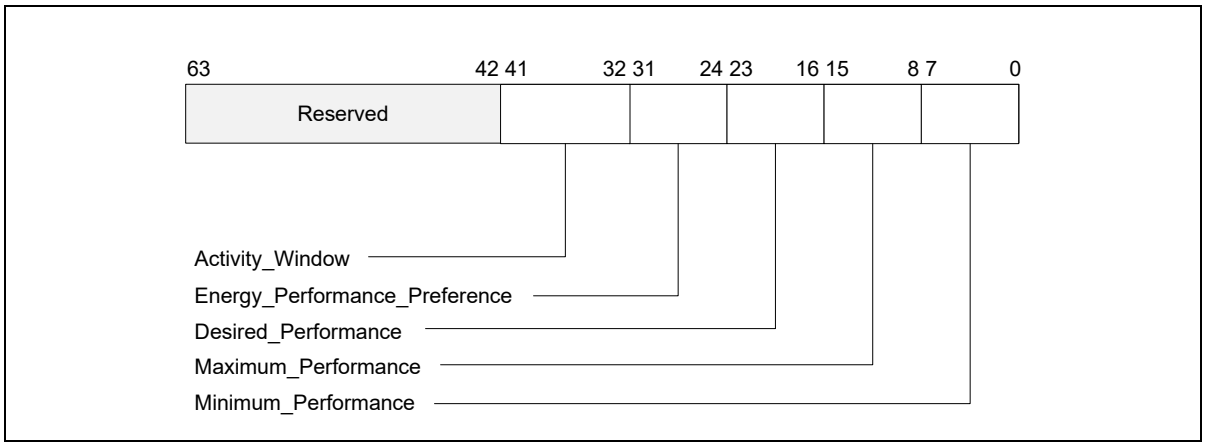


Figure 14-7. IA32_HWP_REQUEST_PKG Register

The structure of the IA32_HWP_REQUEST_PKG MSR (package-level) is identical to the IA32_HWP_REQUEST MSR with the exception of the the Package Control bit field and the five valid bit fields, which do not exist in the IA32_HWP_REQUEST_PKG MSR. Field values written to this MSR apply to all logical processors within the physical package with the exception of logical processors whose IA32_HWP_REQUEST.Package Control field is clear (zero). Single P-state Control mode is only supported when IA32_HWP_REQUEST_PKG is not supported.

14.4.4.3 IA32_HWP_PECI_REQUEST_INFO MSR (Address 775H Package Scope)

When an embedded system controller is integrated in the platform, it can override some of the OS HWP Request settings via the PECI mechanism. PECI initiated settings take precedence over the relevant fields in the IA32_HWP_REQUEST MSR and in the IA32_HWP_REQUEST_PKG MSR, irrespective of the Package Control bit or the Valid Bit values described above. PECI can independently control each of: Minimum Performance, Maximum Performance and EPP fields. This MSR contains both the PECI induced values and the control bits that indicate whether the embedded controller actually set the processor to use the respective value.

PECI override is supported if CPUID[6].EAX[16] is set.

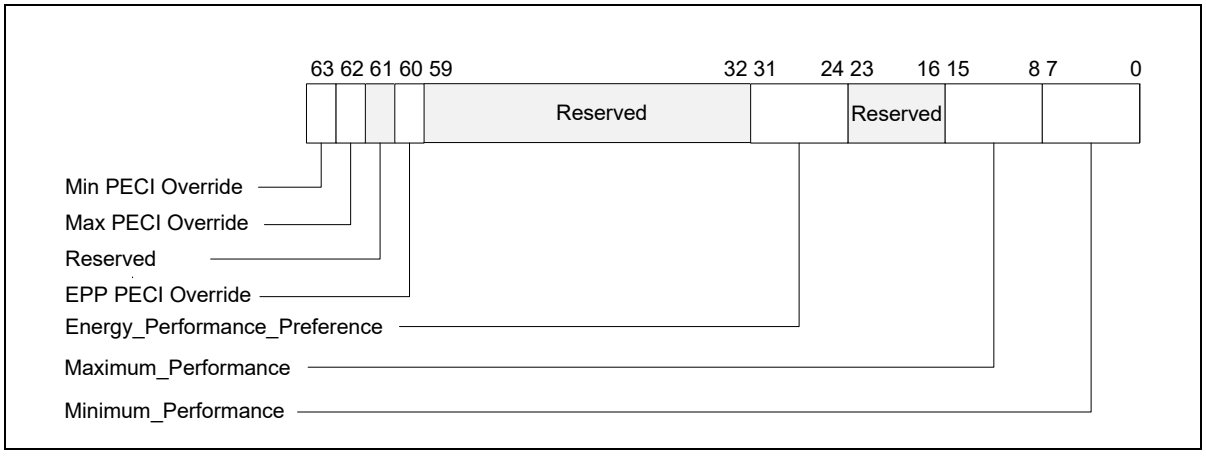


Figure 14-8. IA32_HWP_PECI_REQUEST_INFO MSR

The layout of the IA32_HWP_PECI_REQUEST_INFO MSR is shown in Figure 14-8. This MSR is writable by the embedded controller but is read-only by software executing on the CPU. This MSR has Package scope. The bit fields are described below:

- **Minimum_Performance (bits 7:0, RO)** — Used by the OS to read the latest value of PECI minimum performance input.
- **Maximum_Performance (bits 15:8, RO)** — Used by the OS to read the latest value of PECI maximum performance input.
- Bits 23:16 are reserved and must be zero.
- **Energy_Performance_Preference (bits 31:24, RO)** — Used by the OS to read the latest value of PECI energy performance preference input.
- Bits 59:32 are reserved and must be zero.
- **EPP_PECI_Override (bit 60, RO)** — Indicates whether PECI if currently overriding the Energy Performance Preference input. If set(1), PECI is overriding the Energy Performance Preference input. If clear(0), OS has control over Energy Performance Preference input.
- Bit 61 is reserved and must be zero.
- **Max_PECI_Override (bit 62, RO)** — Indicates whether PECI if currently overriding the Maximum Performance input. If set(1), PECI is overriding the Maximum Performance input. If clear(0), OS has control over Maximum Performance input.
- **Min_PECI_Override (bit 63, RO)** — Indicates whether PECI if currently overriding the Minimum Performance input. If set(1), PECI is overriding the Minimum Performance input. If clear(0), OS has control over Minimum Performance input.

HWP Request Field Hierarchical Resolution

HWP Request field resolution is fed by three MSRs: IA32_HWP_REQUEST, IA32_HWP_REQUEST_PKG and IA32_HWP_PECI_REQUEST_INFO. The flow that the processor goes through to resolve which field value is chosen is shown below.

For each of the two HWP Request fields; Desired and Activity Window:

    If IA32_HWP_REQUEST.PACKAGE_CONTROL = 1 and IA32_HWP_REQUEST.<field> valid bit = 0
        Resolved Field Value = IA32_HWP_REQUEST_PKG.<field>
    Else
        Resolved Field Value = IA32_HWP_REQUEST.<field>

For each of the three HWP Request fields; Min, Max and EPP:

    If IA32_HWP_PECI_REQUEST_INFO.<field> PECI Override bit = 1
        Resolved Field Value = IA32_HWP_PECI_REQUEST_INFO.<field>
    Else if IA32_HWP_REQUEST.PACKAGE_CONTROL = 1 and IA32_HWP_REQUEST.<field> valid bit = 0
        Resolved Field Value = IA32_HWP_REQUEST_PKG.<field>
    Else
        Resolved Field Value = IA32_HWP_REQUEST.<field>

14.4.4.4 IA32_HWP_CTL MSR (Address: 776H Logical Processor Scope)

IA32_HWP_CTL[0] controls the behavior of IA32_HWP_REQUEST Package Control [bit 42]. This control bit allows the IA32_HWP_REQUEST MSR to stay in INIT mode most of the time (Control Bit is equal to its RESET value of 0) thus avoiding actual saving/restoring of the MSR contents when the OS adds it to the register set saved and restored by XSAVES/XRSTORS.

- When IA32_HWP_CTL[0] = 0:
  - If IA32_HWP_REQUEST[42] = 0, the processor ignores all fields of the IA32_HWP_REQUEST_PKG MSR and selects all HWP values (Min, Max, EPP, Desired, Activity Window) from the IA32_HWP_REQUEST MSR.
  - If IA32_HWP_REQUEST[42] = 1, the processor selects the HWP values (Min, Max, EPP, Desired, Activity Window) either from the IA32_HWP_REQUEST MSR or from the IA32_HWP_REQUEST_PKG MSR according

to the values contained in the IA32_HWP_REQUEST MSR bit fields [bits 63:59]. See Section 14.4.4.1 for additional details.

- When IA32_HWP_CTL[0] = 1, the behavior is reversed:
  - If IA32_HWP_REQUEST[42] = 1, the processor ignores all fields of the IA32_HWP_REQUEST_PKG MSR and selects all HWP values (Min, Max, EPP, Desired, Activity Window) from the IA32_HWP_REQUEST MSR.
  - If IA32_HWP_REQUEST[42] = 0, the processor selects the HWP values (Min, Max, EPP, Desired, Activity Window) either from the IA32_HWP_REQUEST MSR or from the IA32_HWP_REQUEST_PKG MSR according to the values contained in the IA32_HWP_REQUEST MSR bit fields [bits 63:59]. See Section 14.4.4.1 for additional details.

Section 14-2 summarizes the IA32_HWP_CTL MSR bit 0 control behavior.

Table 14-2. IA32_HWP_CTL MSR Bit 0 Behavior

Field	Description		
Thread request PKG CTL meaning	Defines which HWP Request MSR is used, whether thread level or package level. When the package MSR is used, the thread MSR valid bits define which thread MSR fields override the package (default 0).		
	IA32_HWP_CTL[PKG_CTL_PLR]	**IA32_HWP_REQUEST[PKG_CTL]**	**HWP Request MSR Used**
	0	0	IA32_HWP_REQUEST MSR
	0	1	IA32_HWP_REQUEST_PKG MSR
	1	0	IA32_HWP_REQUEST_PKG MSR
	1	1	IA32_HWP_REQUEST MSR

This MSR is supported if CPUID[6].EAX[22] is set.

If the IA32_PM_ENABLE[HWP_ENABLE] (bit 0) is not set, access to this MSR will generate a #GP fault.

14.4.5 HWP Feedback

The processor provides several types of feedback to the OS during HWP operation.

The IA32_MPERF MSR and IA32_APERF MSR mechanism (see Section 14.2) allows the OS to calculate the resultant effective frequency delivered over a time period. Energy efficiency and performance optimizations directly impact the resultant effective frequency delivered.

The layout of the IA32_HWP_STATUS MSR is shown in Figure 14-9. It provides feedback regarding changes to IA32_HWP_CAPABILITIES.Guaranteed_Performance, IA32_HWP_CAPABILITIES.Highest_Performance, excursions to IA32_HWP_CAPABILITIES.Minimum_Performance, and PECI_Override entry/exit events. The bit fields are described below:

- **Guaranteed_Performance_Change (bit 0, RWC0)** — If set (1), a change to Guaranteed_Performance has occurred. Software should query IA32_HWP_CAPABILITIES.Guaranteed_Performance value to ascertain the new Guaranteed Performance value and to assess whether to re-adjust HWP hints via IA32_HWP_REQUEST. Software must clear this bit by writing a zero (0).
- Bit 1 is reserved and must be zero.
- **Excursion_To_Minimum (bit 2, RWC0)** — If set (1), an excursion to Minimum_Performance of IA32_HWP_REQUEST has occurred. Software must clear this bit by writing a zero (0).
- **Highest_Change (bit 3, RWC0)** — If set (1), a change to Highest Performance has occurred. Software should query IA32_HWP_CAPABILITIES to ascertain the new Highest Performance value. Software must clear this bit by writing a zero (0). Interrupts upon Highest Performance change are supported if CPUID[6].EAX[15] is set.
- **PECI_Override_Entry (bit 4, RWC0)** — If set (1), an embedded/management controller has started a PECI override of one or more OS control hints (Min, Max, EPP) specified in IA32_HWP_REQUEST or IA32_HWP_REQUEST_PKG. Software may query IA32_HWP_PECI_REQUEST_INFO MSR to ascertain which fields are now overridden via the PECI mechanism and what their values are (see Section 14.4.4.3 for

additional details). Software must clear this bit by writing a zero (0). Interrupts upon PECI override entry are supported if CPUID[6].EAX[16] is set.

- **PECI_Override_Exit (bit 5, RWC0)** — If set (1), an embedded/management controller has stopped overriding one or more OS control hints (Min, Max, EPP) specified in IA32_HWP_REQUEST or IA32_HWP_REQUEST_PKG. Software may query IA32_HWP_PECI_REQUEST_INFO MSR to ascertain which fields are still overridden via the PECI mechanism and which fields are now back under software control (see Section 14.4.4.3 for additional details). Software must clear this bit by writing a zero (0). Interrupts upon PECI override exit are supported if CPUID[6].EAX[16] is set.
- Bits 63:6 are reserved and must be zero.

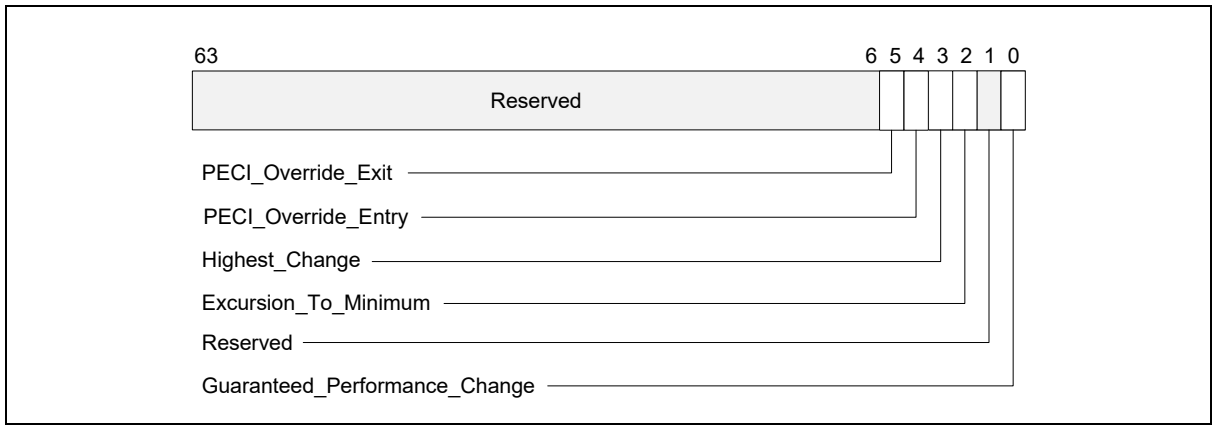


Figure 14-9. IA32_HWP_STATUS MSR

The status bits of IA32_HWP_STATUS must be cleared (0) by software so that a new status condition change will cause the hardware to set the bit again and issue the notification. Status bits are not set for "normal" excursions, e.g., running below Minimum Performance for short durations during C-state exit. Changes to Guaranteed_Performance, Highest_Performance, excursions to Minimum_Performance, or PECI_Override entry/exit will occur no more than once per second.

The OS can determine the specific reasons for a Guaranteed_Performance change or an excursion to Minimum_Performance in IA32_HWP_REQUEST by examining the associated status and log bits reported in the IA32_THERM_STATUS MSR. The layout of the IA32_HWP_STATUS MSR that HWP uses to support software query of HWP feedback is shown in Figure 14-10. The bit fields of IA32_THERM_STATUS associated with HWP feedback are described below (Bit fields of IA32_THERM_STATUS unrelated to HWP can be found in Section 14.8.5.2).

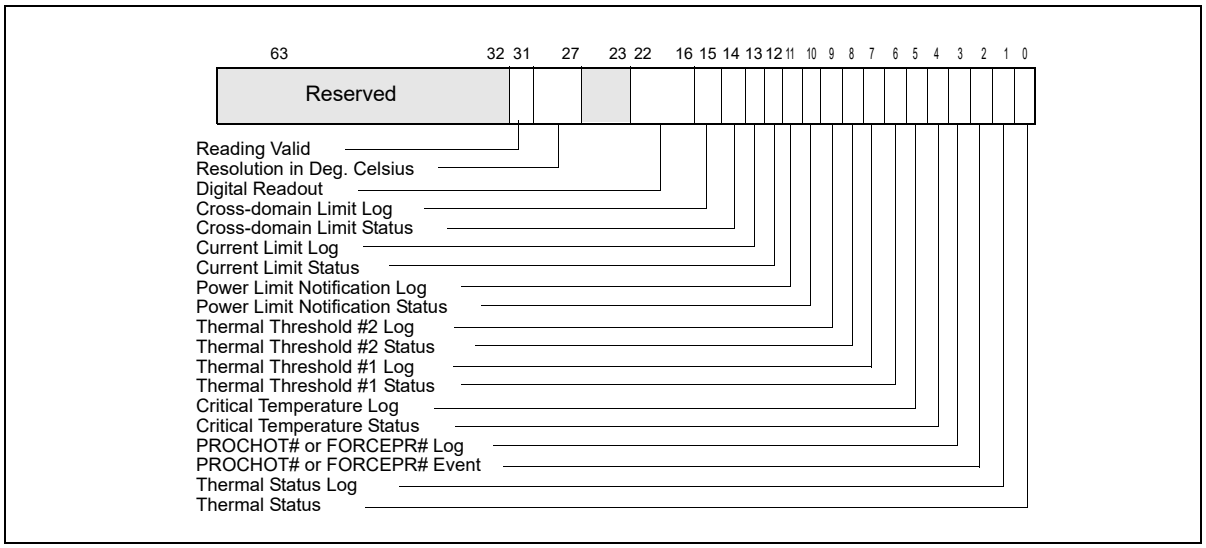


Figure 14-10. IA32_THERM_STATUS Register With HWP Feedback

- Bits 11:0, See Section 14.8.5.2.
- **Current Limit Status (bit 12, RO)** — If set (1), indicates an electrical current limit (e.g. Electrical Design Point/IccMax) is being exceeded and is adversely impacting energy efficiency optimizations.
- **Current Limit Log (bit 13, RWC0)** — If set (1), an electrical current limit has been exceeded that has adversely impacted energy efficiency optimizations since the last clearing of this bit or a reset. This bit is sticky, software may clear this bit by writing a zero (0).
- **Cross-domain Limit Status (bit 14, RO)** — If set (1), indicates another hardware domain (e.g. processor graphics) is currently limiting energy efficiency optimizations in the processor core domain.
- **Cross-domain Limit Log (bit 15, RWC0)** — If set (1), indicates another hardware domain (e.g. processor graphics) has limited energy efficiency optimizations in the processor core domain since the last clearing of this bit or a reset. This bit is sticky, software may clear this bit by writing a zero (0).
- Bits 63:16, See Section 14.8.5.2.

14.4.5.1 Non-Architectural HWP Feedback

The Productive Performance (MSR_PPERF) MSR (non-architectural) provides hardware's view of workload scalability, which is a rough assessment of the relationship between frequency and workload performance, to software. The layout of the MSR_PPERF is shown in Figure 14-11.

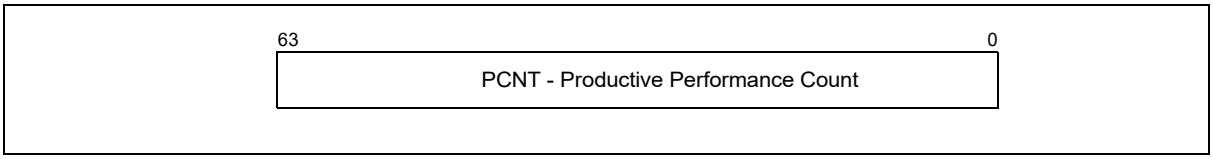


Figure 14-11. MSR_PPERF MSR

- **PCNT (bits 63:0, RO)** — Similar to IA32_APERF but only counts cycles perceived by hardware as contributing to instruction execution (e.g. unhalted and unstalled cycles). This counter increments at the same rate as IA32_APERF, where the ratio of (ΔPCNT/ΔACNT) is an indicator of workload scalability (0% to 100%). Note that values in this register are valid even when HWP is not enabled.

14.4.6 HWP Notifications

Processors may support interrupt-based notification of changes to HWP status as indicated by CPUID. If supported, the IA32_HWP_INTERRUPT MSR is used to enable interrupt-based notifications. Notification events, when enabled, are delivered using the existing thermal LVT entry. The layout of the IA32_HWP_INTERRUPT is shown in Figure 14-12. The bit fields are described below:

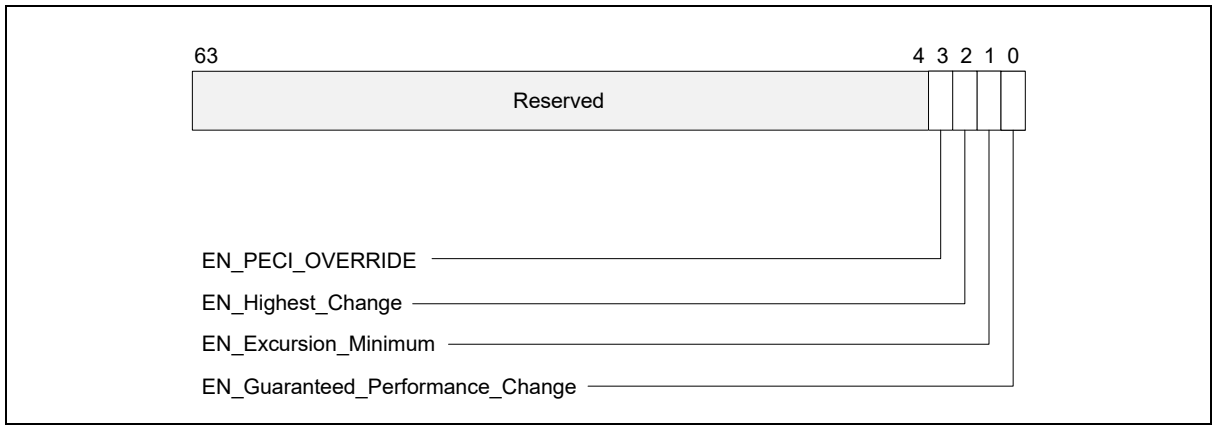


Figure 14-12. IA32_HWP_INTERRUPT MSR

- **EN_Guaranteed_Performance_Change (bit 0, RW)** — When set (1), an HWP Interrupt will be generated whenever a change to the IA32_HWP_CAPABILITIES.Guaranteed_Performance occurs. The default value is 0 (Interrupt generation is disabled).
- **EN_Excursion_Minimum (bit 1, RW)** — When set (1), an HWP Interrupt will be generated whenever the HWP hardware is unable to meet the IA32_HWP_REQUEST.Minimum_Performance setting. The default value is 0 (Interrupt generation is disabled).
- **EN_Highest_Change (bit 2, RW)** — When set (1), an HWP Interrupt will be generated whenever a change to the IA32_HWP_CAPABILITIES.Highest_Performance occurs. The default value is 0 (interrupt generation is disabled). Interrupts upon Highest Performance change are supported if CPUID[6].EAX[15] is set.
- **EN_PECI_OVERRIDE (bit 3, RW)** — When set (1), an HWP Interrupt will be generated whenever PECI starts or stops overriding any of the three HWP fields described in Section 14.4.4.3. The default value is 0 (interrupt generation is disabled). See Section 14.4.5 and Section 14.4.4.3 for details on how the OS learns what is the current set of HWP fields that are overridden by PECI. Interrupts upon PECI override change are supported if CPUID[6].EAX[16] is set.
- Bits 63:4 are reserved and must be zero.

14.4.7 Idle Logical Processor Impact on Core Frequency

Intel processors use one of two schemes for setting core frequency:

1. All cores share same frequency.
2. Each physical core is set to a frequency of its own.

In both cases the two logical processors that share a single physical core are set to the same frequency, so the processor accounts for the IA32_HWP_REQUEST MSR fields of both logical processors when defining the core frequency or the whole package frequency.

When **CPUID[6].EAX[20]** is set and only one logical processor of the two is active, while the other is idle (in any **C1 sub-state** or in a deeper sleep state), only the **active logical processor's** IA32_HWP_REQUEST MSR fields are considered, i.e., the HWP Request fields of a logical processor in the C1E sub-state or in a deeper sleep state are ignored.

Note: when a logical processor is in **C1 state** its HWP Request fields are accounted for.

14.4.8 Fast Write of Uncore MSR (Model Specific Feature)

There are a few logical processor scope MSRs whose values need to be observed outside the logical processor. The WRMSR instruction takes over 1000 cycles to complete (retire) for those MSRs. This overhead forces operating systems to avoid writing them too often whereas in many cases it is preferable that the OS writes them quite frequently for optimal power/performance operation of the processor.

The model specific "Fast Write MSR" feature reduces this overhead by an order of magnitude to a level of 100 cycles for a selected subset of MSRs.

Note: Writes to Fast Write MSRs are posted, i.e., when the WRMSR instruction completes, the data may still be "in transit" within the processor. Software can check the status by querying the processor to ensure data is already visible outside the logical processor (see Section 14.4.8.3 for additional details). Once the data is visible outside the logical processor, software is ensured that later writes by the same logical processor to the same MSR will be visible later (will not bypass the earlier writes).

MSRs that are selected for Fast Write are specified in a special capability MSR (see Section 14.4.8.1). Architectural MSRs that existed prior to the introduction of this feature and are selected for Fast Write, thus turning from slow to fast write MSRs, will be noted as such via a new CPUID bit. New MSRs that are fast upon introduction will be documented as such without an additional CPUID bit.

Three model specific MSRs are associated with the feature itself. They enable *enumerating, controlling and monitoring* it. All three are logical processor scope.

14.4.8.1 FAST_UNCORE_MSRS_CAPABILITY (Address: 0x65F, Logical Processor Scope)

Operating systems or BIOS can read the FAST_UNCORE_MSRS_CAPABILITY MSR to enumerate those MSRs that are Fast Write MSRs.

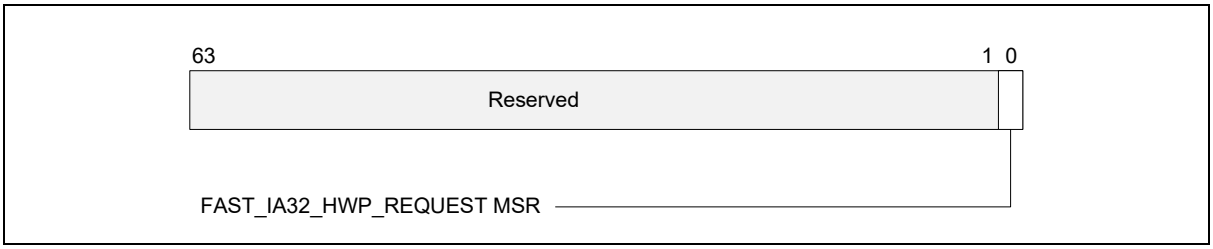


Figure 14-13. FAST_UNCORE_MSRS_CAPABILITY MSR

- **FAST_IA32_HWP_REQUEST MSR (bit 0, RO)** — When set (1), indicates that the IA32_HWP_REQUEST MSR is supported as a Fast Write MSR. A value of 0 indicates the IA32_HWP_REQUEST MSR is not supported as a Fast Write MSR.
- Bits 63:1 are reserved and must be zero.

14.4.8.2 FAST_UNCORE_MSRS_CTL (Address: 0x657, Logical Processor Scope)

Operating Systems or BIOS can use the FAST_UNCORE_MSRS_CTL MSR to opt-in or opt-out for fast write of specific MSRs that are enabled for Fast Write by the processor.

Note: Not all MSRs that are selected for this feature will necessarily have this opt-in/opt-out option. They may be supported in fast write mode only.

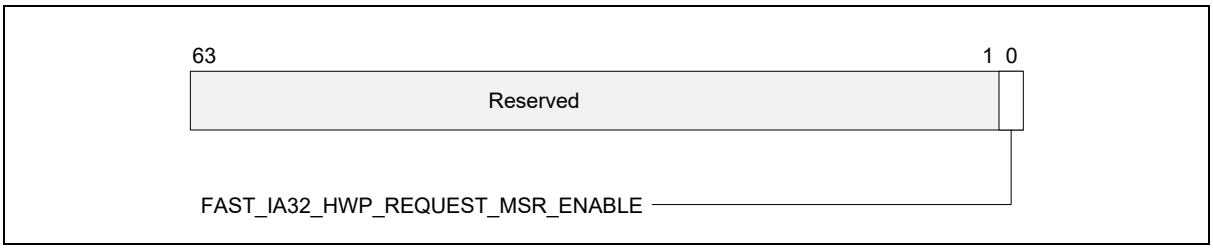


Figure 14-14. FAST_UNCORE_MSRS_CTL MSR

- **FAST_IA32_HWP_REQUEST_MSR_ENABLE (bit 0, RW)** — When set (1), enables fast access mode for the IA32_HWP_REQUEST MSR and sets the low latency, posted IA32_HWP_REQUESRT MSR' CPUID[6].EAX[18]. The default value is 0. Note that this bit can only be enabled once from the default value. Once set, writes to this bit are ignored. Only RESET will clear this bit.
- Bits 63:1 are reserved and must be zero.

14.4.8.3 FAST_UNCORE_MSRS_STATUS (Address: 0x65E, Logical Processor Scope)

Software that executes the WRMSR instruction of a Fast Write MSR can check whether the data is already visible outside the logical processor by reading the FAST_UNCORE_MSRS_STATUS MSR. For each Fast Write MSR there is a status bit that indicates whether the data is already visible outside the logical processor or is still in "transit".

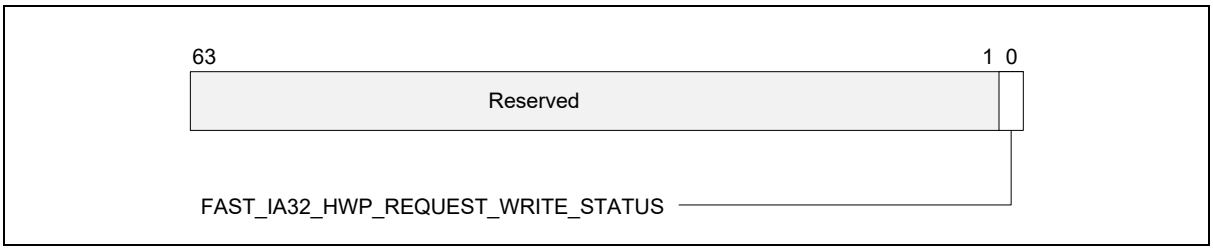


Figure 14-15. FAST_UNCORE_MSRS_STATUS MSR

- **FAST_IA32_HWP_REQUEST_WRITE_STATUS (bit 0, RO)** — 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.
- Bits 63:1 are reserved and must be zero.

14.4.9 Fast_IA32_HWP_REQUEST CPUID

IA32_HWP_REQUEST is an architectural MSR that exists in processors whose CPUID[6].EAX[7] is set (HWP BASE is enabled). This MSR has logical processor scope, but after its contents are written the contents become visible outside the logical processor. When the FAST_IA32_HWP_REQUEST CPUID[6].EAX[18] bit is set, writes to the IA32_HWP_REQUEST MSR are visible outside the logical processor via the "Fast Write" feature described in Section 14.4.8.

14.4.10 Recommendations for OS use of HWP Controls

Common Cases of Using HWP

The default HWP control field values are expected to be suitable for many applications. The OS can enable autonomous HWP for these common cases by

- Setting IA32_HWP_REQUEST.Desired Performance = 0 (hardware autonomous selection determines the performance target). Set IA32_HWP_REQUEST.Activity Window = 0 (enable HW dynamic selection of window size).

To maximize HWP benefit for the common cases, the OS should set

- IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_CAPABILITIES.Lowest_Performance and
- IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Highest_Performance.

Setting IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance is functionally equivalent to using of the IA32_PERF_CTL interface and is therefore not recommended (bypassing HWP).

Calibrating HWP for Application-Specific HWP Optimization

In some applications, the OS may have Quality of Service requirements that may not be met by the default values. The OS can characterize HWP by:

- keeping IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance to prevent non-linearity in the characterization process,
- utilizing the range values enumerated from the IA32_HWP_CAPABILITIES MSR to program IA32_HWP_REQUEST while executing workloads of interest and observing the power and performance result.

The power and performance result of characterization is also influenced by the IA32_HWP_REQUEST.Energy Performance Preference field, which must also be characterized.

Characterization can be used to set IA32_HWP_REQUEST.Minimum_Performance to achieve the required QOS in terms of performance. If IA32_HWP_REQUEST.Minimum_Performance is set higher than IA32_HWP_CAPABILITIES.Guaranteed Performance then notification of excursions to Minimum Performance may be continuous.

If autonomous selection does not deliver the required workload performance, the OS should assess the current delivered effective frequency and for the duration of the specific performance requirement set IA32_HWP_REQUEST.Desired_Performance ≠ 0 and adjust IA32_HWP_REQUEST.Energy_Performance_Preference as necessary to achieve the required workload performance. The MSR_PPERF.PCNT value can be used to better comprehend the potential performance result from adjustments to IA32_HWP_REQUEST.Desired_Performance. The OS should set IA32_HWP_REQUEST.Desired_Performance = 0 to re-enable autonomous selection.

Tuning for Maximum Performance or Lowest Power Consumption

Maximum performance will be delivered by setting IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Highest_Performance and setting IA32_HWP_REQUEST.Energy_Performance_Preference = 0 (performance preference).

Lowest power will be achieved by setting IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Lowest_Performance and setting IA32_HWP_REQUEST.Energy_Performance_Preference = 0FFH (energy efficiency preference).

Mixing Logical Processor and Package Level HWP Field Settings

Using the IA32_HWP_REQUEST Package_Control bit and the five valid bits in that MSR, the OS can mix and match between selecting the Logical Processor scope fields and the Package level fields. For example, the OS can set all logical cores' IA32_HWP_REQUEST.Package_Control bit to '1', and for those logical processors if it prefers a different EPP value than the one set in the IA32_HWP_REQUEST_PKG MSR, the OS can set the desired EPP value and the EPP valid bit. This overrides the package EPP value for only a subset of the logical processors in the package.

Additional Guidelines

Set IA32_HWP_REQUEST.Energy_Performance_Preference as appropriate for the platform's current mode of operation. For example, a mobile platforms' setting may be towards performance preference when on AC power and more towards energy efficiency when on DC power.

The use of the Running Average Power Limit (RAPL) processor capability (see section 14.7.1) is highly recommended when HWP is enabled. Use of IA32_HWP_Request.Maximum_Performance for thermal control is subject to limitations and can adversely impact the performance of other processor components e.g. Graphics

If default values deliver undesirable performance latency in response to events, the OS should set IA32_HWP_REQUEST. Activity_Window to a low (non-zero) value and IA32_HWP_REQUEST.Energy_Performance_Preference towards performance (0) for the event duration.

Similarly, for "real-time" threads, set IA32_HWP_REQUEST.Energy_Performance_Preference towards performance (0) and IA32_HWP_REQUEST. Activity_Window to a low value, e.g. 01H, for the duration of their execution.

When executing low priority work that may otherwise cause the hardware to deliver high performance, set IA32_HWP_REQUEST. Activity_Window to a longer value and reduce the IA32_HWP_Request.Maximum_Performance value as appropriate to control energy efficiency. Adjustments to IA32_HWP_REQUEST.Energy_Performance_Preference may also be necessary.

14.5 HARDWARE DUTY CYCLING (HDC)

Intel processors may contain support for Hardware Duty Cycling (HDC), which enables the processor to autonomously force its components inside the physical package into idle state. For example, the processor may selectively force only the processor cores into an idle state.

HDC is disabled by default on processors that support it. System software can dynamically enable or disable HDC to force one or more components into an idle state or wake up those components previously forced into an idle state. Forced Idling (and waking up) of multiple components in a physical package can be done with one WRMSR to a packaged-scope MSR from any logical processor within the same package.

HDC does not delay events such as timer expiration, but it may affect the latency of short (less than 1 msec) software threads, e.g. if a thread is forced to idle state just before completion and entering a "natural idle".

HDC forced idle operation can be thought of as operating at a lower effective frequency. The effective average frequency computed by software will include the impact of HDC forced idle.

The primary use of HDC is enable system software to manage low active workloads to increase the package level C6 residency. Additionally, HDC can lower the effective average frequency in case or power or thermal limitation.

When HDC forces a logical processor, a processor core or a physical package to enter an idle state, its C-State is set to C3 or deeper. The deep "C-states" referred to in this section are processor-specific C-states.

14.5.1 Hardware Duty Cycling Programming Interfaces

The programming interfaces provided by HDC include the following:

- The CPUID instruction allows software to discover the presence of HDC support in an Intel processor. Specifically, execute CPUID instruction with EAX=06H as input, bit 13 of EAX indicates the processor's support of the following aspects of HDC.
  - — Availability of HDC baseline resource, CPUID.06H:EAX[bit 13]: If this bit is set, HDC provides the following architectural MSRs: IA32_PKG_HDC_CTL, IA32_PM_CTL1, and the IA32_THREAD_STALL MSRs.

- Additionally, HDC may provide several non-architectural MSR.

Table 14-3. Architectural and non-Architecture MSRs Related to HDC

Address	Architectural	Register Name	Description
DB0H	Y	IA32_PKG_HDC_CTL	Package Enable/Disable HDC.
DB1H	Y	IA32_PM_CTL1	Per-logical-processor select control to allow/block HDC forced idling.
DB2H	Y	IA32_THREAD_STALL	Accumulate stalled cycles on this logical processor due to HDC forced idling.
653H	N	MSR_CORE_HDC_RESIDENCY	Core level stalled cycle counter due to HDC forced idling on one or more logical processor.
655H	N	MSR_PKG_HDC_SHALLOW_RESIDENCY	Accumulate the cycles the package was in C2[1] state and at least one logical processor was in forced idle
656H	N	MSR_PKG_HDC_DEEP_RESIDENCY	Accumulate the cycles the package was in the software specified Cx[1] state and at least one logical processor was in forced idle. Cx is specified in MSR_PKG_HDC_CONFIG_CTL.
652H	N	MSR_PKG_HDC_CONFIG_CTL	HDC configuration controls

NOTES:

1. The package "C-states" referred to in this section are processor-specific C-states.

14.5.2 Package level Enabling HDC

The layout of the IA32_PKG_HDC_CTL MSR is shown in Figure 14-16. IA32_PKG_HDC_CTL is a writable MSR from any logical processor in a package. The bit fields are described below:

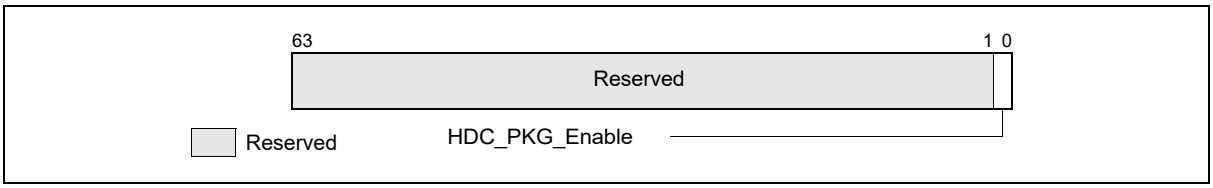


Figure 14-16. IA32_PKG_HDC_CTL MSR

- **HDC_PKG_Enable (bit 0, R/W)** — Software sets this bit to enable HDC operation by allowing the processor to force to idle all "HDC-allowed" (see Figure 14.5.3) logical processors in the package. Clearing this bit disables HDC operation in the package by waking up all the processor cores that were forced into idle by a previous '0'-to-'1' transition in IA32_PKG_HDC_CTL.HDC_PKG_Enable. This bit is writable only if CPUID.06H:EAX[bit 13] = 1. Default = zero (0).
- Bits 63:1 are reserved and must be zero.

After processor support is determined via CPUID, system software can enable HDC operation by setting IA32_PKG_HDC_CTL.HDC_PKG_Enable to 1. At reset, IA32_PKG_HDC_CTL.HDC_PKG_Enable is cleared to 0. A '0'-to-'1' transition in HDC_PKG_Enable allows the processor to force to idle all HDC-allowed (indicated by the non-zero state of IA32_PM_CTL1[bit 0]) logical processors in the package. A '1'-to-'0' transition wakes up those HDC force-idled logical processors.

Software can enable or disable HDC using this package level control multiple times from any logical processor in the package. Note the latency of writing a value to the package-visible IA32_PKG_HDC_CTL.HDC_PKG_Enable is longer than the latency of a WRMSR operation to a Logical Processor MSR (as opposed to package level MSR) such as: IA32_PM_CTL1 (described in Section 14.5.3). Propagation of the change in IA32_PKG_HDC_CTL.HDC_PKG_Enable and reaching all HDC idled logical processor to be woken up may take on the order of core C6 exit latency.

14.5.3 Logical-Processor Level HDC Control

The layout of the IA32_PM_CTL1 MSR is shown in Figure 14-17. Each logical processor in a package has its own IA32_PM_CTL1 MSR. The bit fields are described below:

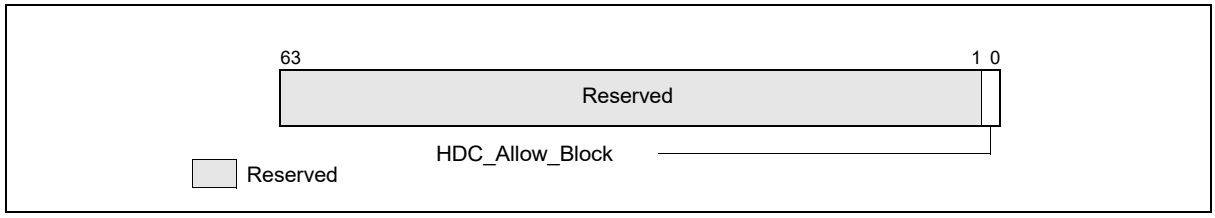


Figure 14-17. IA32_PM_CTL1 MSR

- **HDC_Allow_Block (bit 0, R/W)** — Software sets this bit to allow this logical processors to honor the package-level IA32_PKG_HDC_CTL.HDC_PKG_Enable control. Clearing this bit prevents this logical processor from using the HDC. This bit is writable only if CPUID.06H:EAX[bit 13] = 1. Default = one (1).
- Bits 63:1 are reserved and must be zero.

Fine-grain OS control of HDC operation at the granularity of per-logical-processor is provided by IA32_PM_CTL1. At RESET, all logical processors are allowed to participate in HDC operation such that OS can manage HDC using the package-level IA32_PKG_HDC_CTL.

Writes to IA32_PM_CTL1 complete with the latency that is typical to WRMSR to a Logical Processor level MSR. When the OS chooses to manage HDC operation at per-logical-processor granularity, it can write to IA32_PM_CTL1 on one or more logical processors as desired. Each write to IA32_PM_CTL1 must be done by code that executes on the logical processor targeted to be allowed into or blocked from HDC operation.

Blocking one logical processor for HDC operation may have package level impact. For example, the processor may decide to stop duty cycling of all other Logical Processors as well.

The propagation of IA32_PKG_HDC_CTL.HDC_PKG_Enable in a package takes longer than a WRMSR to IA32_PM_CTL1. The last completed write to IA32_PM_CTL1 on a logical processor will be honored when a '0'-to-'1' transition of IA32_PKG_HDC_CTL.HDC_PKG_Enable arrives to a logical processor.

14.5.4 HDC Residency Counters

There is a collection of counters available for software to track various residency metrics related to HDC operation. In general, HDC residency time is defined as the time in HDC forced idle state at the granularity of per-logical-processor, per-core, or package. At the granularity of per-core/package-level HDC residency, at least one of the logical processor in a core/package must be in the HDC forced idle state.

14.5.4.1 IA32_THREAD_STALL

Software can track per-logical-processor HDC residency using the architectural MSR IA32_THREAD_STALL.The layout of the IA32_THREAD_STALL MSR is shown in Figure 14-18. Each logical processor in a package has its own IA32_THREAD_STALL MSR. The bit fields are described below:

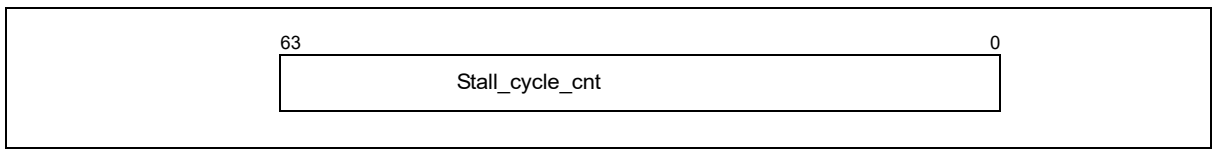


Figure 14-18. IA32_THREAD_STALL MSR

- **Stall_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after the logical processor exits from the forced idled C-state. At each update, the number of cycles that the logical processor was stalled due to forced-idle will be added to the counter. This counter is available only if CPUID.06H:EAX[bit 13] = 1. Default = zero (0).

A value of zero in IA32_THREAD_STALL indicates either HDC is not supported or the logical processor never serviced any forced HDC idle. A non-zero value in IA32_THREAD_STALL indicates the HDC forced-idle residency times of the logical processor. It also indicates the forced-idle cycles due to HDC that could appear as C0 time to traditional OS accounting mechanisms (e.g. time-stamping OS idle/exit events).

Software can read IA32_THREAD_STALL irrespective of the state of IA32_PKG_HDC_CTL and IA32_PM_CTL1, as long as CPUID.06H:EAX[bit 13] = 1.

14.5.4.2 Non-Architectural HDC Residency Counters

Processors that support HDC operation may provide the following model-specific HDC residency counters.

MSR_CORE_HDC_RESIDENCY

Software can track per-core HDC residency using the counter MSR_CORE_HDC_RESIDENCY. This counter increments when the core is in C3 state or deeper (all logical processors in this core are idle due to either HDC or other mechanisms) and at least one of the logical processors is in HDC forced idle state. The layout of the MSR_CORE_HDC_RESIDENCY is shown in Figure 14-19. Each processor core in a package has its own MSR_CORE_HDC_RESIDENCY MSR. The bit fields are described below:

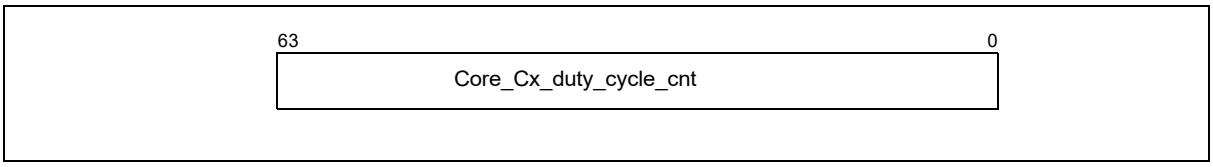


Figure 14-19. MSR_CORE_HDC_RESIDENCY MSR

- **Core_Cx_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after core C-state exit from a forced idled C-state. At each update, the increment counts cycles when the core is in a Cx state (all its logical processor are idle) and at least one logical processor in this core was forced into idle state due to HDC. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR will cause a #GP fault. Default = zero (0).

A value of zero in MSR_CORE_HDC_RESIDENCY indicates either HDC is not supported or this processor core never serviced any forced HDC idle.

MSR_PKG_HDC_SHALLOW_RESIDENCY

The counter MSR_PKG_HDC_SHALLOW_RESIDENCY allows software to track HDC residency time when the package is in C2 state, all processor cores in the package are not active and at least one logical processor was forced into idle state due to HDC. The layout of the MSR_PKG_HDC_SHALLOW_RESIDENCY is shown in Figure 14-20. There is one MSR_PKG_HDC_SHALLOW_RESIDENCY per package. The bit fields are described below:

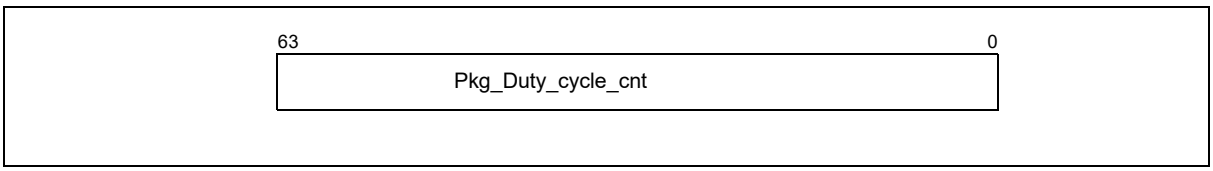


Figure 14-20. MSR_PKG_HDC_SHALLOW_RESIDENCY MSR

- **Pkg_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. Package shallow residency may be implementation specific. In the initial implementation, the threshold is package C2-state. The count is updated only after package C2-state exit from a forced idled C-state. At each update, the increment counts cycles when the package is in C2 state and at least one processor core in this package was forced into idle state due to HDC. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).

A value of zero in MSR_PKG_HDC_SHALLOW_RESIDENCY indicates either HDC is not supported or this processor package never serviced any forced HDC idle.

MSR_PKG_HDC_DEEP_RESIDENCY

The counter MSR_PKG_HDC_DEEP_RESIDENCY allows software to track HDC residency time when the package is in a software-specified package Cx state, all processor cores in the package are not active and at least one logical processor was forced into idle state due to HDC. Selection of a specific package Cx state can be configured using MSR_PKG_HDC_CONFIG. The layout of the MSR_PKG_HDC_DEEP_RESIDENCY is shown in Figure 14-21. There is one MSR_PKG_HDC_DEEP_RESIDENCY per package. The bit fields are described below:

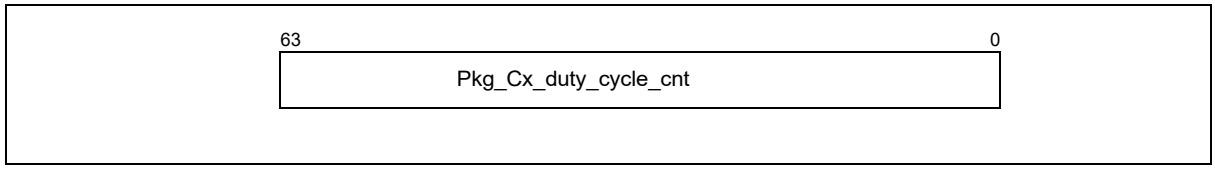


Figure 14-21. MSR_PKG_HDC_DEEP_RESIDENCY MSR

- **Pkg_Cx_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after package C-state exit from a forced idle state. At each update, the increment counts cycles when the package is in the software-configured Cx state and at least one processor core in this package was forced into idle state due to HDC. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).

A value of zero in MSR_PKG_HDC_SHALLOW_RESIDENCY indicates either HDC is not supported or this processor package never serviced any forced HDC idle.

MSR_PKG_HDC_CONFIG

MSR_PKG_HDC_CONFIG allows software to configure the package Cx state that the counter MSR_PKG_HDC_DEEP_RESIDENCY monitors. The layout of the MSR_PKG_HDC_CONFIG is shown in Figure 14-22. There is one MSR_PKG_HDC_CONFIG per package. The bit fields are described below:

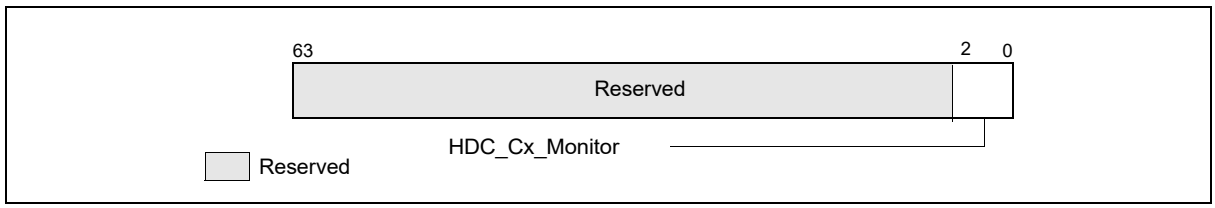


Figure 14-22. MSR_PKG_HDC_CONFIG MSR

- **Pkg_Cx_Monitor (bits 2:0, R/W)** — Selects which package C-state the MSR_HDC_DEEP_RESIDENCY counter will monitor. The encoding of the HDC_Cx_Monitor field are: **0**: no-counting; **1**: count package C2 only, **2**: count package C3 and deeper; **3**: count package C6 and deeper; **4**: count package C7 and deeper; other encodings are reserved. If CPUID.06H:EAX[bit 13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).
- Bits 63:3 are reserved and must be zero.

14.5.5 MPERF and APERF Counters Under HDC

HDC operation can be thought of as an average effective frequency drop due to all or some of the Logical Processors enter an idle state period.

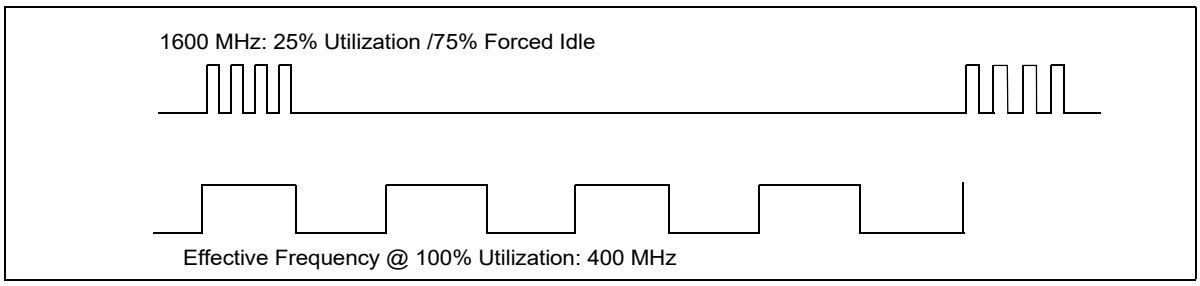


Figure 14-23. Example of Effective Frequency Reduction and Forced Idle Period of HDC

By default, the IA32_MPERF counter counts during forced idle periods as if the logical processor was active. The IA32_APERF counter does not count during forced idle state. This counting convention allows the OS to compute the average effective frequency of the Logical Processor between the last MWAIT exit and the next MWAIT entry (OS visible C0) by ΔACNT/ΔMCNT * TSC Frequency.

14.6 HARDWARE FEEDBACK INTERFACE AND INTEL® THREAD DIRECTOR

Intel processors that enumerate CPUID.06H.0H:EAX.HW_FEEDBACK[bit 19] as 1 support Hardware Feedback Interface (HFI). Hardware provides guidance to the Operating System (OS) scheduler to perform optimal workload scheduling through a hardware feedback interface structure in memory. Details on this table structure are described in Section 14.6.1.

Intel processors that enumerate CPUID.06H.0H:EAX[bit 23] as 1 support Intel® Thread Director. Hardware provides guidance to the Operating System (OS) scheduler to perform optimal workload scheduling through a memory resident table and software thread specific index (Class ID) that points into that table and selects which data to use for that software thread. Details on this table structure are described in Section 14.6.2.

14.6.1 Hardware Feedback Interface Table Structure

This structure has a global header that is 16 bytes in size. Following this global header, there is one 8 byte entry per logical processor in the socket. The structure is designed as follows.

Table 14-4. Hardware Feedback Interface Structure

Byte Offset	Size (Bytes)	Description
0	16	Global Header
16	8	Per Logical Processor Entry
24	8	Per Logical Processor Entry
…	…	…
16 + n*8	8	Per Logical Processor Entry

The global header is structured as shown in Table 14-5.

Table 14-5. Hardware Feedback Interface Global Header Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	8	Timestamp	Timestamp of when the table was last updated by hardware. This is a timestamp in crystal clock units. Initialized by OS to 0.
8	1	Performance Capability Changed	If set to 1, indicates the performance capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
9	1	Energy Efficiency Capability Changed	If set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
10	6	Reserved	Initialized by OS to 0.

The per logical processor scheduler feedback entry is structured as follows. The operating system can determine the index of the logical processor feedback entry for a logical processor using CPUID.06H.0H:EDX[31:16] by executing CPUID on that logical processor.

Table 14-6. Hardware Feedback Interface Logical Processor Entry Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	1	Performance Capability	Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback. CPUID.06H.0H:EDX[0] enumerates support for Performance capability reporting.
1	1	Energy Efficiency Capability	Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback. CPUID.06H.0H:EDX[1] enumerates support for Energy Efficiency capability reporting.
2	6	Reserved	OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback.

14.6.2 Intel® Thread Director Table Structure

This structure has a global header that is at least 16 bytes in size. Its size depends on the number of classes and capabilities enumerated by the CPUID instruction (see notes below Table 14-7). Following this global header there are multiple Logical Processor related entries. The structure is designed as follows.

Table 14-7. Intel® Thread Director Table Structure

Byte Offset[1,2,3]	Size (Bytes)	Description
0	8 + CP[4]*CL[4] + R8[5]	Global Header
8 + CP*CL + R8	CL*CP + R8	Per Logical Processor $Entry_0$[6]
8 + 2*(CP*CL + R8)	CL*CP + R8	Per Logical Processor $Entry_1$
…	…	…
8 + (N^7 -1)*(CP*CL + R8)	CL*CP +R8	Logical Processor $Entry_{N-1}$

NOTES:

1. Byte offset of $Capability_{cp}$ of $Class_{cl}$ change indication: 8 + CP * cl + cp.
2. Byte offset of LP $Entry_i$: 8 + (i+1) * (CP * CL + R8).
3. Byte offset of $capability_{cp}$ of $class_{cl}$ of LP $Entry_i$: 8 + (i+1) * (CP * CL + R8) + CP * cl + cp.
4. Both upper case CL and CP denote total number of classes and capabilities defined for the processor. Lower case cl and cp denote one instance of a class or capability. cl and cp are counted starting at zero. See "CPUID—CPU Identification" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A for the number of classes (CL) and the number of supported capabilities (CP). CP (# of capabilities): number of enumerated bits in CPUID.06H.0H.EDX[7:0] and CL (# of classes): CPUID.06H.0H.ECX[15:8].
5. R8 is the number of bytes necessary to round up the Capability Change Indication array and the Logical Processor Entry to whole multiple of 8 bytes.
6. Table size: 8 + (N+1)* (CP * CL + R8).
7. N is the number of Logical Processor Entries in the table. It is not greater than the number of Logical Processors on the socket, but may be lower.
8. The Operating System can determine the index for the Logical Processor Entry within the Intel Thread Director table using CPUID.06H.0H:EDX[31:16] by executing the CPUID instruction on that Logical Processor.
9. The Operating System should allocate space to accommodate for one such structure per socket in the system.
10. The Intel Thread Director table structure extends the Hardware Feedback Interface table structure without breaking backward compatibility. The Hardware Feedback Interface can be viewed as having two capabilities and a single class.

The global header is structured as shown in Table 14-8.

Table 14-8. Intel® Thread Director Global Header Structure

Byte Offset	Size (Bytes)	Description	
0	8	Time-stamp of when the table was last updated by hardware. This is a time-stamp in crystal clock units. Initialized by OS to 0.	
8	1	Class 0 Performance Capability Changed	If set to 1, indicates the performance capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
8 + 1	1	Class 0 Energy Efficiency Capability Changed	If set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
…			
8 + CP - 1	1	Class 0 change indication for Capability #(CP-1) if exists	Unavailable for capabilities that are not enumerated.
8 + CP	1	Class 1 Performance Capability Changed	If set to 1, indicates the performance capability field for one or more logical processors was updated in the table. Initialized by OS to 0.

Table 14-8. Intel® Thread Director Global Header Structure (Contd.)

Byte Offset	Size (Bytes)	Description	
8 + CP + 1	1	Class 1 Energy Efficiency Capability Changed	If set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
...			
8 + 2*CP - 1	1	Class 1 change indication for Capability #(CP-1) if exists	Unavailable for capabilities that are not enumerated.
...			Change indication for Capabilities of additional Classes if exist.
8 + (CL-1)*CP	1	Class #(CL-1) Performance Capability Changed	If set to 1, indicates the performance capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
8 + (CL-1)*CP + 1	1	Class #(CL-1) Energy Efficiency Capability Changed	If set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table. Initialized by OS to 0.
...			
8 + CL*CP - 1	1	Class #(CL-1) change indication for Capability #(CP-1) if exists	Unavailable for capabilities that are not enumerated.
8 + CL*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by OS to 0 prior to enabling Intel Thread Director.

The logical processor capability entry in the Intel Thread Director table is structured as follows.

Table 14-9. Intel® Thread Director Logical Processor Entry Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	1	Performance Capability	Class 0 Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by OS to 0.
1	1	Energy Efficiency Capability	Class 0 Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. Initialized by OS to 0.
...			
CP - 1	1	Capability #(CP-1)	Class 0 Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).
CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by OS to 0 prior to enabling Intel Thread Director.
CP + R8	1	Performance Capability	Class 1 Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by OS to 0.
CP + 1	1	Energy Efficiency Capability	Class 1 Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. Initialized by OS to 0.

Table 14-9. Intel® Thread Director Logical Processor Entry Structure (Contd.)

Byte Offset	Size (Bytes)	Field Name	Description
...			
2*CP - 1	1	Capability #(CP-1)	Class 1 Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).
2*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by OS to 0 prior to enabling Intel Thread Director.
...			
(CL-1)*CP	1	Performance Capability	Class #(CL-1) Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by OS to 0.
(CL-1)*CP + 1	1	Energy Efficiency Capability	Class #(CL-1) Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. Initialized by OS to 0.
...			
CL*CP - 1	1	Capability #(CP-1)	Class #(CL-1) Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).
CL*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by OS to 0 prior to enabling Intel Thread Director.

14.6.3 Intel® Thread Director Usage Model

When the OS Scheduler needs to decide which one of multiple free logical processors to assign to a software thread that is ready to execute, it can choose one of the following options:

1. The free logical processor with the highest performance value of that software thread class, if the system is scheduling for performance.
2. The free logical processor with the highest energy efficiency value of that software thread class, if the system is scheduling for energy efficiency.

When the OS Scheduler needs to decide which of two logical processors (i,j) to assign to which of two software threads whose Class IDs are k1 and k2, it can compute the two performance ratios: Perf $Ratio_1$ = $Perf_{ik1}$ / $Perf_{jk1}$ and Perf $Ratio_2$ = $Perf_{ik2}$ / $Perf_{jk2}$, or two energy efficiency ratios: Energy Eff. $Ratio_1$ = Energy Eff_{ik1} / Energy Eff_{jk1} and Energy Eff. $Ratio_2$ = Energy Eff_{ik2} / Energy Eff_{jk2} between the two logical processors for each of the two classes, depending on whether the OS is scheduling for performance or for energy efficiency.

For example, assume that the system is scheduling for performance and that Perf $Ratio_1$ > Perf $Ratio_2$. The OS Scheduler will assign the software thread whose Class ID is k1 to logical processor i, and the one whose Class ID is k2 to logical processor j.

When the two software threads in question belong to the same Class ID, the OS Scheduler can schedule to higher performance logical processors within that class when scheduling for performance and to higher energy efficiency logical processors within that class when scheduling for energy efficiency.

The highest to lowest ordering may be different between classes across cores and between the performance column and the energy efficiency column of the same class across cores.

14.6.4 Hardware Feedback Interface Pointer

The physical address of the HFI/Intel Thread Director structure is programmed by the OS into a package scoped MSR named IA32_HW_FEEDBACK_PTR. The MSR is structured as follows:

- Bits 63:MAXPHYADDR[1] – Reserved.
- Bits MAXPHYADDR-1:12 – ADDR. This is the physical address of the page frame of the first page of this structure.
- Bits 11:1 – Reserved.
- Bit 0 – Valid. When set to 1, indicates a valid pointer is programmed into the ADDR field of the MSR.

The address of this MSR is 17D0H. This MSR is cleared on processor reset to its default value of 0. It retains its value upon INIT.

CPUID.06H.0H:EDX[11:8] enumerates the size of memory that must be allocated by the OS for this structure.

14.6.5 Hardware Feedback Interface Configuration

The operating system enables HFI/Intel Thread Director using a package scoped MSR named IA32_HW_FEEDBACK_CONFIG (address 17D1H). This MSR is cleared on processor reset to its default value of 0. It retains its value upon INIT.

The MSR is structured as follows:

- Bits 63:2 – Reserved.
- Bit 1 – Enable Intel Thread Director (or multi-class support). Both bits 0 and 1 must be set for Intel Thread Director to be enabled. The extra class columns in the Intel Thread Director table are updated by hardware immediately following setting those two bits, as well as during run time as necessary.
- Bit 0 – Enable. When set to 1, enables HFI.

Before enabling HFI, the OS must set a valid hardware feedback interface structure using IA32_HW_FEEDBACK_PTR.

When the OS sets bit 0 only, the hardware populates class 0 capabilities only in the HFI structure. When bit 1 is set after or together with bit 0, the Intel Thread Director multi-class structure is populated.

When either the HFI structure or the Intel Thread Director structure are ready to use by the OS, the hardware sets IA32_PACKAGE_THERM_STATUS[bit 26]. An interrupt is generated by the hardware if IA32_PACKAGE_THERM_INTERRUPT[bit 25] is set.

When the OS clears bit 1 but leaves bit 0 set, Intel Thread Director is disabled, but HFI is kept operational. IA32_PACKAGE_THERM_STATUS[bit 26] is NOT set in this case.

Clearing bit 0 disables both HFI and Intel Thread Director, independent of the bit 1 state. Setting bit 1 to '1' while keeping bit 0 at '0' is an invalid combination which is quietly ignored.

When the OS clears bit 0, hardware sets the IA32_PACKAGE_THERM_STATUS[bit 26] to 1 to acknowledge disabling of the interface. The OS should wait for this bit to be set to 1 to reclaim the memory of the Intel Thread Director structure, as by setting IA32_PACKAGE_THERM_STATUS[bit 26] hardware guarantees not to write into the Intel Thread Director structure anymore.

The OS may clear bit 0 only after receiving an indication from the hardware that the structure initialization is complete via the same IA32_PACKAGE_THERM_STATUS[bit 26], following enabling of HFI/Intel Thread Director, thus avoiding a race condition between OS and hardware.

Bit 1 is valid only if CPUID[6].EAX[bit 23] is set. When setting this bit while support is not enumerated, the hardware generates #GP.

Table 14-10 summarizes the control options described above.

See Section 14.6.9 for details on scenarios where IA32_HW_FEEDBACK_CONFIG bits are implicitly reset by the hardware.

1. MAXPHYADDR is reported in CPUID.80000008H:EAX[7:0].

Table 14-10. IA32_HW_FEEDBACK_CONFIG Control Options

Pre-Bit 1	Pre-Bit 0	Post-Bit 1	Post-Bit 0	Action	IA32_PACKAGE_THERM_STATUS [bit 26] and Interrupt
0	0	0	0	Reset value.	Both Hardware Feedback Interface and Intel Thread Director are disabled, no status bit set, no interrupt is generated.
0	0	0	1	Enable HFI structure.	Set the status bit and generate interrupt if enabled.
0	0	1	0	Invalid option; quietly ignored by the hardware.	No action (no update in the table).
0	0	1	1	Enable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
0	1	0	0	Disable HFI support.	Set the status bit and generate interrupt if enabled.
0	1	1	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
0	1	1	1	Enable Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	0	0	0	No action; keeps HFI and Intel Thread Director disabled.	No action (no update in the table).
1	0	0	1	Enable HFI.	Set the status bit and generate interrupt if enabled.
1	0	1	1	Enable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	1	0	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	1	0	1	Disable Intel Thread Director; keep HFI enabled.	No action (no update in the table).
1	1	1	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.

14.6.6 Hardware Feedback Interface Notifications

The IA32_PACKAGE_THERM_STATUS MSR is extended with a new bit, hardware feedback interface structure change status (bit 26, R/WC0), to indicate that the hardware has updated the HFI/Intel Thread Director structure. This is a sticky bit and once set, indicates that the OS should read the structure to determine the change and adjust its scheduling decisions. Once set, the hardware will not generate any further updates to this structure until the OS clears this bit by writing 0.

The OS can enable interrupt-based notifications when the structure is updated by hardware through a new enable bit, hardware feedback interrupt enable (bit 25, R/W), in the IA32_PACKAGE_THERM_INTERRUPT MSR. When this bit is set to 1, it enables the generation of an interrupt when the HFI/Intel Thread Director structure is updated by hardware. When the enable bit transitions from 0 to 1, hardware will generate an initial notify, with the IA32_PACKAGE_THERM_STATUS bit 26 set to 1, to indicate that the OS should read the current HFI/Intel Thread Director structure.

14.6.7 Hardware Feedback Interface and Intel® Thread Director Structure Dynamic Update

The HFI/Intel Thread Director structure can be updated dynamically during run time. Changes to the structure may occur to one or more of its cells. Such changes may occur for one or more logical processors. The hardware sets a non zero value in the "capability change" field of the HFI/Intel Thread Director structure as an indication for the OS to read that capability for all logical processors. A thermal interrupt is delivered to indicate to the OS that the structure has just changed. Section 14.6.6 contains more details on this notification mechanism. The hardware clears all "capability change" fields after the OS resets IA32_PACKAGE_THERM_STATUS[bit 26].

Zeroing a performance or energy efficiency cell hints to the OS that it is beneficial not to schedule software threads of that class on the associated logical processor for performance or energy efficiency reasons, respectively. If SMT is supported, it may be the case that the hardware zeroes one of the core's logical processors only. Zeroing the performance and energy efficiency cells of all classes for a logical processor implies that the hardware provides a hint to the OS to completely avoid scheduling work on that logical processor.

A few example reasons for runtime changes in the HGS/Intel Thread Director Table:

- Over clocking run time update that changes the capability values.
- Change in run time physical constraints.
- Run time performance or energy efficiency optimization.
- Change in core frequency, voltage, or power budget.

14.6.8 Logical Processor Scope Intel® Thread Director Configuration

The operating system enables Intel Thread Director at the logical processor scope using a logical processor scope MSR named IA32_HW_FEEDBACK_THREAD_CONFIG (address 17D4H).

The MSR is read/write and is structured as follows:

- Bits 63:1 – Reserved.
- Bit 0 – Enables Intel Thread Director. When set to 1, logical processor scope Intel Thread Director is enabled. Default is 0 (disabled).

Bit 0 of the logical processor scope configuration MSR can be cleared or set regardless of the state of the HFI/Intel Thread Director package configuration MSR state. Even when bit 0 of all logical processor configuration MSRs is clear, the processor can still update the Intel Thread Director structure if it is still enabled in the IA32_HW_FEEDBACK_CONFIG package scope MSR. When the operating system clears IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0], hardware clears the history accumulated on that logical processor which otherwise drives assigning the Class ID to the software thread that executes on that logical processor. As long as IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] is set, the Class ID is available for the operating system to read, independent of the state of the package scope IA32_HW_FEEDBACK_CONFIG[1:0] bits.

See Section 14.6.9 for details on scenarios where IA32_HW_FEEDBACK_CONFIG bits are implicitly reset by the hardware.

14.6.9 Implicit Reset of Package and Logical Processor Scope Configuration MSRs

HFI/Intel Thread Director enable bits are reset by hardware in the following scenarios:

1. When GETSEC[SENTER] is executed:
   a. The processor implicitly resets the HFI/Intel Thread Director enable bits in the IA32_HW_FEEDBACK_CONFIG MSR on all sockets (packages) in the system.
   b. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on all logical processors in the system across all sockets.
   c. The processor implicitly clears the HFI/Intel Thread Director table structure pointer in the IA32_HW_FEEDBACK_PTR package MSR across all sockets.
2. When GETSEC[ENTERACCS] is executed:

a. The processor implicitly resets the HFI/Intel Thread Director enable bits in the IA32_HW_FEEDBACK_CONFIG MSR on the socket where the GETSEC[ENTERACCS] instruction was executed.

b. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on all logical processors on the socket where the GETSEC[ENTERACCS] instruction was executed.

c. The processor implicitly clears the HFI/Intel Thread Director table structure pointer in the IA32_HW_FEEDBACK_PTR package MSR on the socket where the GETSEC[ENTERACCS] instruction was executed.

3. When an INIT or a wait-for-SIPI state are processed by a logical processor:

a. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on that logical processor, whether the signal was in the context of GETSEC[ENTERACCS] or not.

If the OS requires HFI/Intel Thread Director to be active after exiting the measured environment or when processing a SIPI event, it should re-enable HFI/Intel Thread Director.

14.6.10 Logical Processor Scope Intel® Thread Director Run Time Characteristics

The processor provides the operating system with run time feedback about the execution characteristics of the software thread executing on logical processors whose IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] is set.

The run time feedback is communicated via a read-only MSR named IA32_THREAD_FEEDBACK_CHAR. This is a logical processor scope MSR whose address is 17D2H. This MSR is structured as follows:

- Bit 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.
- Bits 62:8 – Reserved.
- Bits 7:0 – Application Class ID, pointing into the Intel Thread Director structure described in Table 14-8.

This MSR is valid only if CPUID.06H:EAX[bit 23] is set.

The valid bit is cleared by the hardware in the following cases:

- The hardware does not have enough information to provide the operating system with a reliable Class ID.
- The operating system cleared the logical processor's IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] bit.

The HRESET instruction is executed while configured to reset the Intel Thread Director history.

14.6.11 Logical Processor Scope History

The operating system can reset the Intel Thread Director related history accumulated on the current logical processor it is executing on by issuing the HRESET instruction. See "CPUID—CPU Identification" in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A* for enumeration of the HRESET instruction. See also the "HRESET — History Reset" instruction description in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

14.6.11.1 Enabling Intel® Thread Director History Reset

The IA32_HRESET_ENABLE MSR is a read/write MSR and is structured as follows:

- Bits 63:32 – Reserved.
- Bits 31:1 – Reserved for other capabilities that can be reset by the HRESET instruction.
- Bit 0 – Enable reset of the Intel Thread Director history.

The operating system should set IA32_HRESET_ENABLE[bit 0] to enable Intel Thread Director history reset via the HRESET instruction.

14.6.11.2 Implicit Intel® Thread Director History Reset

The Intel Thread Director history is implicitly reset in the following scenarios:

1. When the processor enters or exits SMM mode and IA32_DEBUGCTL MSR.FREEZE_WHILE_SMM (bit 14) is set, the Intel Thread Director history is implicitly reset by the processor.
2. When GETSEC[SENTER] is executed, the processor resets the Intel Thread Director history on all logical processors in the system, including logical processors on other sockets (other than the one GETSEC[SENTER] is executed).
3. When GETSEC[ENTERACCS] is executed, the processor resets the Intel Thread Director history on the logical processor it is executed on.
4. When an INIT or a wait-for-SIPI state are processed by a logical processor, the Intel Thread Director history is reset whether the signal was a result of GETSEC[ENTERACCS] or not.

If the operating system requires HFI/Intel Thread Director to be active after exiting the measured environment or when processing a SIPI event, it should re-enable HFI/Intel Thread Director.

14.7 MWAIT EXTENSIONS FOR ADVANCED POWER MANAGEMENT

IA-32 processors may support a number of C-states[1] that reduce power consumption for inactive states. Intel Core Solo and Intel Core Duo processors support both deeper C-state and MWAIT extensions that can be used by OS to implement power management policy.

Software should use CPUID to discover if a target processor supports the enumeration of MWAIT extensions. If CPUID.05H.ECX[Bit 0] = 1, the target processor supports MWAIT extensions and their enumeration (see Chapter 4, "Instruction Set Reference, M-U," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*).

If CPUID.05H.ECX[Bit 1] = 1, the target processor supports using interrupts as break-events for MWAIT, even when interrupts are disabled. Use this feature to measure C-state residency as follows:

- Software can write to bit 0 in the MWAIT Extensions register (ECX) when issuing an MWAIT to enter into a processor-specific C-state or sub C-state.
- When a processor comes out of an inactive C-state or sub C-state, software can read a timestamp before an interrupt service routine (ISR) is potentially executed.

CPUID.05H.EDX allows software to enumerate processor-specific C-states and sub C-states available for use with MWAIT extensions. IA-32 processors may support more than one C-state of a given C-state type. These are called sub C-states. Numerically higher C-state have higher power savings and latency (upon entering and exiting) than lower-numbered C-state.

At CPL = 0, system software can specify desired C-state and sub C-state by using the MWAIT hints register (EAX). Processors will not go to C-state and sub C-state deeper than what is specified by the hint register. If CPL > 0 and if MONITOR/MWAIT is supported at CPL > 0, the processor will only enter C1-state (regardless of the C-state request in the hints register).

Executing MWAIT generates an exception on processors operating at a privilege level where MONITOR/MWAIT are not supported.

NOTE

If MWAIT is used to enter a C-state (including sub C-state) that is numerically higher than C1, a store to the address range armed by MONITOR instruction will cause the processor to exit MWAIT if the store was originated by other processor agents. A store from non-processor agent may not cause the processor to exit MWAIT.

1. The processor-specific C-states defined in MWAIT extensions can map to ACPI defined C-state types (C0, C1, C2, C3). The mapping relationship depends on the definition of a C-state by processor implementation and is exposed to OSPM by the BIOS using the ACPI defined _CST table.

14.8 THERMAL MONITORING AND PROTECTION

The IA-32 architecture provides the following mechanisms for monitoring temperature and controlling thermal power:

1. The **catastrophic shutdown detector** forces processor execution to stop if the processor's core temperature rises above a preset limit.
2. **Automatic and adaptive thermal monitoring mechanism**s force the processor to reduce it's power consumption in order to operate within predetermined temperature limits.
3. The **software controlled clock modulation mechanism** permits operating systems to implement power management policies that reduce power consumption; this is in addition to the reduction offered by automatic thermal monitoring mechanisms.
4. **On-die digital thermal sensor and interrupt mechanisms** permit the OS to manage thermal conditions natively without relying on BIOS or other system board components.

The first mechanism is not visible to software. The other three mechanisms are visible to software using processor feature information returned by executing CPUID with EAX = 1.

The second mechanism includes:

- **Automatic thermal monitoring** provides two modes of operation. One mode modulates the clock duty cycle; the second mode changes the processor's frequency. Both modes are used to control the core temperature of the processor.
- **Adaptive thermal monitoring** can provide flexible thermal management on processors made of multiple cores.

The third mechanism modulates the clock duty cycle of the processor. As shown in Figure 14-24, the phrase 'duty cycle' does not refer to the actual duty cycle of the clock signal. Instead it refers to the time period during which the clock signal is allowed to drive the processor chip. By using the stop clock mechanism to control how often the processor is clocked, processor power consumption can be modulated.

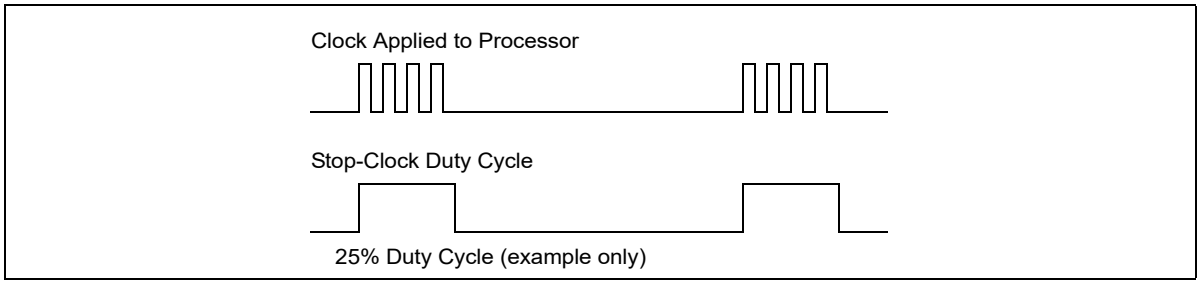


Figure 14-24. Processor Modulation Through Stop-Clock Mechanism

For previous automatic thermal monitoring mechanisms, software controlled mechanisms that changed processor operating parameters to impact changes in thermal conditions. Software did not have native access to the native thermal condition of the processor; nor could software alter the trigger condition that initiated software program control.

The fourth mechanism (listed above) provides access to an on-die digital thermal sensor using a model-specific register and uses an interrupt mechanism to alert software to initiate digital thermal monitoring.

14.8.1 Catastrophic Shutdown Detector

P6 family processors introduced a thermal sensor that acts as a catastrophic shutdown detector. This catastrophic shutdown detector was also implemented in Pentium 4, Intel Xeon and Pentium M processors. It is always enabled. When processor core temperature reaches a factory preset level, the sensor trips and processor execution is halted until after the next reset cycle.

14.8.2 Thermal Monitor

Pentium 4, Intel Xeon and Pentium M processors introduced a second temperature sensor that is factory-calibrated to trip when the processor's core temperature crosses a level corresponding to the recommended thermal design envelop. The trip-temperature of the second sensor is calibrated below the temperature assigned to the catastrophic shutdown detector.

14.8.2.1 Thermal Monitor 1

The Pentium 4 processor uses the second temperature sensor in conjunction with a mechanism called Thermal Monitor 1 (TM1) to control the core temperature of the processor. TM1 controls the processor's temperature by modulating the duty cycle of the processor clock. Modulation of duty cycles is processor model specific. Note that the processors STPCLK# pin is not used here; the stop-clock circuitry is controlled internally.

Support for TM1 is indicated by CPUID.1:EDX.TM[bit 29] = 1.

TM1 is enabled by setting the thermal-monitor enable flag (bit 3) in IA32_MISC_ENABLE [see Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*]. Following a power-up or reset, the flag is cleared, disabling TM1. BIOS is required to enable only one automatic thermal monitoring modes. Operating systems and applications must not disable the operation of these mechanisms.

14.8.2.2 Thermal Monitor 2

An additional automatic thermal protection mechanism, called Thermal Monitor 2 (TM2), was introduced in the Intel Pentium M processor and also incorporated in newer models of the Pentium 4 processor family. Intel Core Duo and Solo processors, and Intel Core 2 Duo processor family all support TM1 and TM2. TM2 controls the core temperature of the processor by reducing the operating frequency and voltage of the processor and offers a higher performance level for a given level of power reduction than TM1.

TM2 is triggered by the same temperature sensor as TM1. The mechanism to enable TM2 may be implemented differently across various IA-32 processor families with different CPUID signatures in the family encoding value, but will be uniform within an IA-32 processor family.

Support for TM2 is indicated by CPUID.1:ECX.TM2[bit 8] = 1.

14.8.2.3 Two Methods for Enabling TM2

On processors with CPUID family/model/stepping signature encoded as 0x69n or 0x6Dn (early Pentium M processors), TM2 is enabled if the TM_SELECT flag (bit 16) of the MSR_THERM2_CTL register is set to 1 (Figure 14-25) and bit 3 of the IA32_MISC_ENABLE register is set to 1.

Following a power-up or reset, the TM_SELECT flag may be cleared. BIOS is required to enable either TM1 or TM2. Operating systems and applications must not disable mechanisms that enable TM1 or TM2. If bit 3 of the IA32_MISC_ENABLE register is set and TM_SELECT flag of the MSR_THERM2_CTL register is cleared, TM1 is enabled.

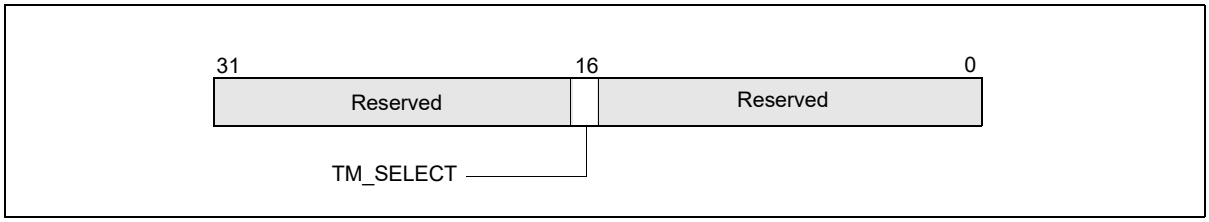


Figure 14-25. MSR_THERM2_CTL Register On Processors with CPUID Family/Model/Stepping Signature Encoded as 0x69n or 0x6Dn

On processors introduced after the Pentium 4 processor (this includes most Pentium M processors), the method used to enable TM2 is different. TM2 is enable by setting bit 13 of IA32_MISC_ENABLE register to 1. This applies to Intel Core Duo, Core Solo, and Intel Core 2 processor family.

The target operating frequency and voltage for the TM2 transition after TM2 is triggered is specified by the value written to MSR_THERM2_CTL, bits 15:0 (Figure 14-26). Following a power-up or reset, BIOS is required to enable at least one of these two thermal monitoring mechanisms. If both TM1 and TM2 are supported, BIOS may choose to enable TM2 instead of TM1. Operating systems and applications must not disable the mechanisms that enable TM1or TM2; and they must not alter the value in bits 15:0 of the MSR_THERM2_CTL register.

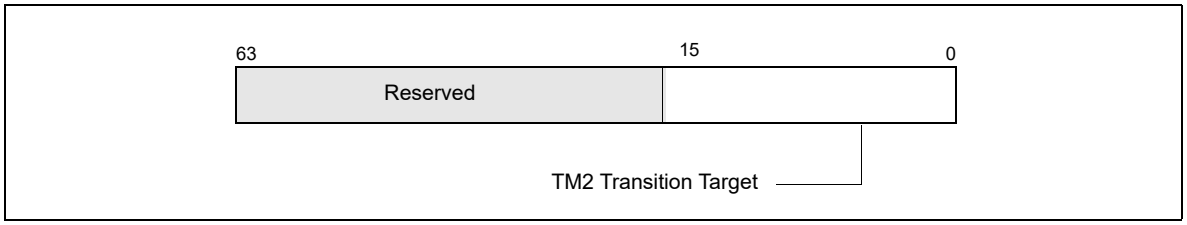


Figure 14-26. MSR_THERM2_CTL Register for Supporting TM2

14.8.2.4 Performance State Transitions and Thermal Monitoring

If the thermal control circuitry (TCC) for thermal monitor (TM1/TM2) is active, writes to the IA32_PERF_CTL will effect a new target operating point as follows:

- If TM1 is enabled and the TCC is engaged, the performance state transition can commence before the TCC is disengaged.
- If TM2 is enabled and the TCC is engaged, the performance state transition specified by a write to the IA32_PERF_CTL will commence after the TCC has disengaged.

14.8.2.5 Thermal Status Information

The status of the temperature sensor that triggers the thermal monitor (TM1/TM2) is indicated through the thermal status flag and thermal status log flag in the IA32_THERM_STATUS MSR (see Figure 14-27).

The functions of these flags are:

- **Thermal Status flag, bit 0** — When set, indicates that the processor core temperature is currently at the trip temperature of the thermal monitor and that the processor power consumption is being reduced via either TM1 or TM2, depending on which is enabled. When clear, the flag indicates that the core temperature is below the thermal monitor trip temperature. This flag is read only.
- **Thermal Status Log flag, bit 1** — When set, indicates that the thermal sensor has tripped since the last power-up or reset or since the last time that software cleared this flag. This flag is a sticky bit; once set it remains set until cleared by software or until a power-up or reset of the processor. The default state is clear.

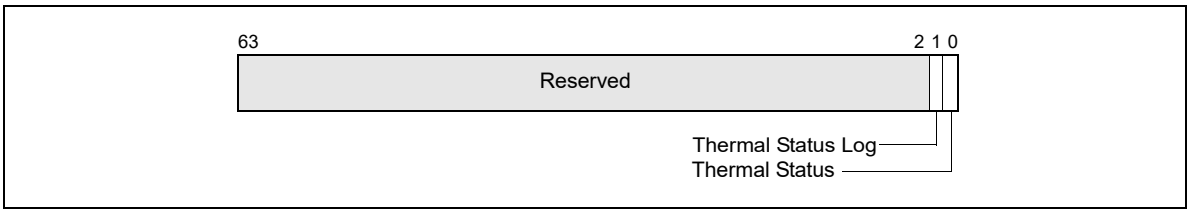


Figure 14-27. IA32_THERM_STATUS MSR

After the second temperature sensor has been tripped, the thermal monitor (TM1/TM2) will remain engaged for a minimum time period (on the order of 1 ms). The thermal monitor will remain engaged until the processor core temperature drops below the preset trip temperature of the temperature sensor, taking hysteresis into account.

While the processor is in a stop-clock state, interrupts will be blocked from interrupting the processor. This holding off of interrupts increases the interrupt latency, but does not cause interrupts to be lost. Outstanding interrupts remain pending until clock modulation is complete.

The thermal monitor can be programmed to generate an interrupt to the processor when the thermal sensor is tripped; this is called a thermal interrupt. The delivery mode, mask and vector for this interrupt can be programmed through the thermal entry in the local APIC's LVT (see Section 10.5.1, "Local Vector Table"). The low-temperature interrupt enable and high-temperature interrupt enable flags in the IA32_THERM_INTERRUPT MSR (see Figure 14-28) control when the interrupt is generated; that is, on a transition from a temperature below the trip point to above and/or vice-versa.

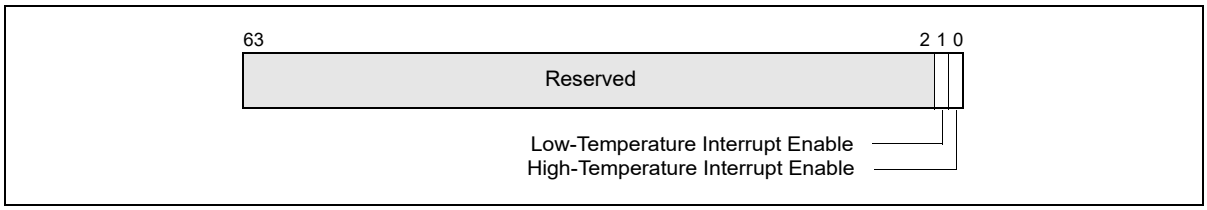


Figure 14-28. IA32_THERM_INTERRUPT MSR

- **High-Temperature Interrupt Enable flag, bit 0** — Enables an interrupt to be generated on the transition from a low-temperature to a high-temperature when set; disables the interrupt when clear.(R/W).
- **Low-Temperature Interrupt Enable flag, bit 1** — Enables an interrupt to be generated on the transition from a high-temperature to a low-temperature when set; disables the interrupt when clear.

The thermal interrupt can be masked by the thermal LVT entry. After a power-up or reset, the low-temperature interrupt enable and high-temperature interrupt enable flags in the IA32_THERM_INTERRUPT MSR are cleared (interrupts are disabled) and the thermal LVT entry is set to mask interrupts. This interrupt should be handled either by the operating system or system management mode (SMM) code.

Note that the operation of the thermal monitoring mechanism has no effect upon the clock rate of the processor's internal high-resolution timer (time stamp counter).

14.8.2.6 Adaptive Thermal Monitor

The Intel Core 2 Duo processor family supports enhanced thermal management mechanism, referred to as Adaptive Thermal Monitor (Adaptive TM).

Unlike TM2, Adaptive TM is not limited to one TM2 transition target. During a thermal trip event, Adaptive TM (if enabled) selects an optimal target operating point based on whether or not the current operating point has effectively cooled the processor.

Similar to TM2, Adaptive TM is enable by BIOS. The BIOS is required to test the TM1 and TM2 feature flags and enable all available thermal control mechanisms (including Adaptive TM) at platform initiation.

Adaptive TM is available only to a subset of processors that support TM2.

In each chip-multiprocessing (CMP) silicon die, each core has a unique thermal sensor that triggers independently. These thermal sensor can trigger TM1 or TM2 transitions in the same manner as described in Section 14.8.2.1 and Section 14.8.2.2. The trip point of the thermal sensor is not programmable by software since it is set during the fabrication of the processor.

Each thermal sensor in a processor core may be triggered independently to engage thermal management features. In Adaptive TM, both cores will transition to a lower frequency and/or lower voltage level if one sensor is triggered.

Triggering of this sensor is visible to software via the thermal interrupt LVT entry in the local APIC of a given core.

14.8.3 Software Controlled Clock Modulation

Pentium 4, Intel Xeon and Pentium M processors also support software-controlled clock modulation. This provides a means for operating systems to implement a power management policy to reduce the power consumption of the

processor. Here, the stop-clock duty cycle is controlled by software through the IA32_CLOCK_MODULATION MSR (see Figure 14-29).

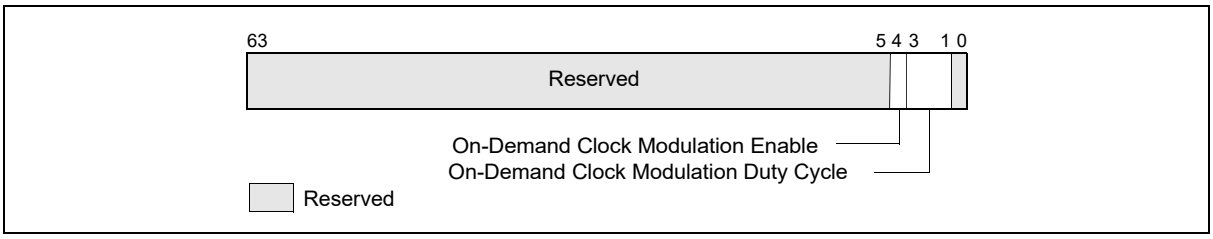


Figure 14-29. IA32_CLOCK_MODULATION MSR

The IA32_CLOCK_MODULATION MSR contains the following flag and field used to enable software-controlled clock modulation and to select the clock modulation duty cycle:

- **On-Demand Clock Modulation Enable, bit 4** — Enables on-demand software controlled clock modulation when set; disables software-controlled clock modulation when clear.
- **On-Demand Clock Modulation Duty Cycle, bits 1 through 3** — Selects the on-demand clock modulation duty cycle (see Table 14-11). This field is only active when the on-demand clock modulation enable flag is set.

Note that the on-demand clock modulation mechanism (like the thermal monitor) controls the processor's stop-clock circuitry internally to modulate the clock signal. The STPCLK# pin is not used in this mechanism.

Table 14-11. On-Demand Clock Modulation Duty Cycle Field Encoding

Duty Cycle Field Encoding	Duty Cycle
000B	Reserved
001B	12.5% (Default)
010B	25.0%
011B	37.5%
100B	50.0%
101B	63.5%
110B	75%
111B	87.5%

The on-demand clock modulation mechanism can be used to control processor power consumption. Power management software can write to the IA32_CLOCK_MODULATION MSR to enable clock modulation and to select a modulation duty cycle. If on-demand clock modulation and TM1 are both enabled and the thermal status of the processor is hot (bit 0 of the IA32_THERM_STATUS MSR is set), clock modulation at the duty cycle specified by TM1 takes precedence, regardless of the setting of the on-demand clock modulation duty cycle.

For Hyper-Threading Technology enabled processors, the IA32_CLOCK_MODULATION register is duplicated for each logical processor. In order for the On-demand clock modulation feature to work properly, the feature must be enabled on all the logical processors within a physical processor. If the programmed duty cycle is not identical for all the logical processors, the processor core clock will modulate to the highest duty cycle programmed for processors with any of the following CPUID DisplayFamily_DisplayModel signatures (see CPUID instruction in Chapter3, "Instruction Set Reference, A-L" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*): 06_1A, 06_1C, 06_1E, 06_1F, 06_25, 06_26, 06_27, 06_2C, 06_2E, 06_2F, 06_35, 06_36, and 0F_xx. For all other processors, if the programmed duty cycle is not identical for all logical processors in the same core, the processor core will modulate at the lowest programmed duty cycle.

For multiple processor cores in a physical package, each processor core can modulate to a programmed duty cycle independently.

For the P6 family processors, on-demand clock modulation was implemented through the chipset, which controlled clock modulation through the processor's STPCLK# pin.

14.8.3.1 Extension of Software Controlled Clock Modulation

Extension of the software controlled clock modulation facility supports on-demand clock modulation duty cycle with 4-bit dynamic range (increased from 3-bit range). Granularity of clock modulation duty cycle is increased to 6.25% (compared to 12.5%).

Four bit dynamic range control is provided by using bit 0 in conjunction with bits 3:1 of the IA32_CLOCK_MODULATION MSR (see Figure 14-30).

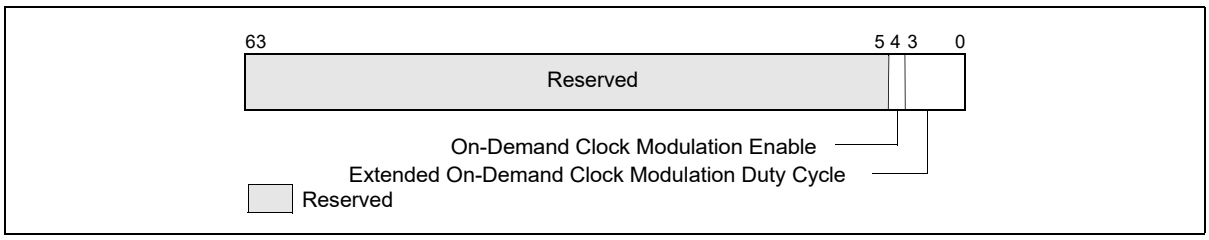


Figure 14-30. IA32_CLOCK_MODULATION MSR with Clock Modulation Extension

Extension to software controlled clock modulation is supported only if CPUID.06H:EAX[Bit 5] = 1. If CPUID.06H:EAX[Bit 5] = 0, then bit 0 of IA32_CLOCK_MODULATION is reserved.

14.8.4 Detection of Thermal Monitor and Software Controlled Clock Modulation Facilities

The ACPI flag (bit 22) of the CPUID feature flags indicates the presence of the IA32_THERM_STATUS, IA32_THERM_INTERRUPT, IA32_CLOCK_MODULATION MSRs, and the xAPIC thermal LVT entry.

The TM1 flag (bit 29) of the CPUID feature flags indicates the presence of the automatic thermal monitoring facilities that modulate clock duty cycles.

14.8.4.1 Detection of Software Controlled Clock Modulation Extension

Processor's support of software controlled clock modulation extension is indicated by CPUID.06H:EAX[Bit 5] = 1.

14.8.5 On Die Digital Thermal Sensors

On die digital thermal sensor can be read using an MSR (no I/O interface). In Intel Core Duo processors, each core has a unique digital sensor whose temperature is accessible using an MSR. The digital thermal sensor is the preferred method for reading the die temperature because (a) it is located closer to the hottest portions of the die, (b) it enables software to accurately track the die temperature and the potential activation of thermal throttling.

14.8.5.1 Digital Thermal Sensor Enumeration

The processor supports a digital thermal sensor if CPUID.06H.EAX[0] = 1. If the processor supports digital thermal sensor, EBX[bits 3:0] determine the number of thermal thresholds that are available for use.

Software sets thermal thresholds by using the IA32_THERM_INTERRUPT MSR. Software reads output of the digital thermal sensor using the IA32_THERM_STATUS MSR.

14.8.5.2 Reading the Digital Sensor

Unlike traditional analog thermal devices, the output of the digital thermal sensor is a temperature relative to the maximum supported operating temperature of the processor.

Temperature measurements returned by digital thermal sensors are always at or below TCC activation temperature. Critical temperature conditions are detected using the "Critical Temperature Status" bit. When this bit is set, the processor is operating at a critical temperature and immediate shutdown of the system should occur. Once the "Critical Temperature Status" bit is set, reliable operation is not guaranteed.

See Figure 14-31 for the layout of IA32_THERM_STATUS MSR. Bit fields include:

- **Thermal Status (bit 0, RO)** — This bit indicates whether the digital thermal sensor high-temperature output signal (PROCHOT#) is currently active. Bit 0 = 1 indicates the feature is active. This bit may not be written by software; it reflects the state of the digital thermal sensor.
- **Thermal Status Log (bit 1, R/WC0)** — This is a sticky bit that indicates the history of the thermal sensor high temperature output signal (PROCHOT#). Bit 1 = 1 if PROCHOT# has been asserted since a previous RESET or the last time software cleared the bit. Software may clear this bit by writing a zero.
- **PROCHOT# or FORCEPR# Event (bit 2, RO)** — Indicates whether PROCHOT# or FORCEPR# is being asserted by another agent on the platform.

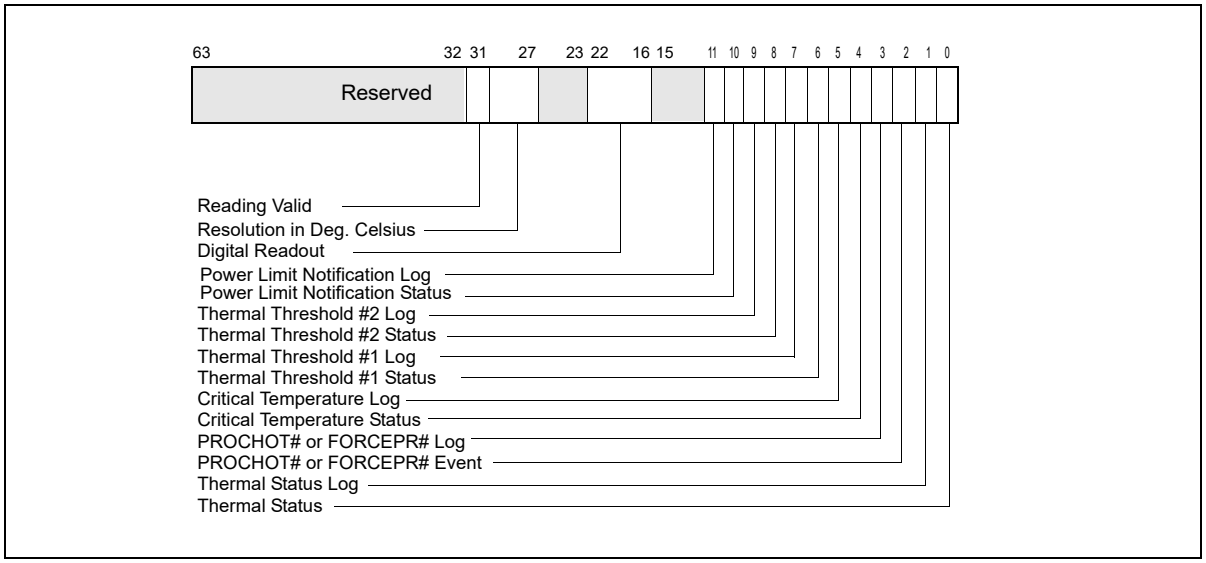


Figure 14-31. IA32_THERM_STATUS Register

- **PROCHOT# or FORCEPR# Log (bit 3, R/WC0)** — Sticky bit that indicates whether PROCHOT# or FORCEPR# has been asserted by another agent on the platform since the last clearing of this bit or a reset. If bit 3 = 1, PROCHOT# or FORCEPR# has been externally asserted. Software may clear this bit by writing a zero. External PROCHOT# assertions are only acknowledged if the Bidirectional Prochot feature is enabled.
- **Critical Temperature Status (bit 4, RO)** — Indicates whether the critical temperature detector output signal is currently active. If bit 4 = 1, the critical temperature detector output signal is currently active.
- **Critical Temperature Log (bit 5, R/WC0)** — Sticky bit that indicates whether the critical temperature detector output signal has been asserted since the last clearing of this bit or reset. If bit 5 = 1, the output signal has been asserted. Software may clear this bit by writing a zero.
- **Thermal Threshold #1 Status (bit 6, RO)** — Indicates whether the actual temperature is currently higher than or equal to the value set in Thermal Threshold #1. If bit 6 = 0, the actual temperature is lower. If bit 6 = 1, the actual temperature is greater than or equal to TT#1. Quantitative information of actual temperature can be inferred from Digital Readout, bits 22:16.
- **Thermal Threshold #1 Log (bit 7, R/WC0)** — Sticky bit that indicates whether the Thermal Threshold #1 has been reached since the last clearing of this bit or a reset. If bit 7 = 1, the Threshold #1 has been reached. Software may clear this bit by writing a zero.
- **Thermal Threshold #2 Status (bit 8, RO)** — Indicates whether actual temperature is currently higher than or equal to the value set in Thermal Threshold #2. If bit 8 = 0, the actual temperature is lower. If bit 8 = 1, the

actual temperature is greater than or equal to TT#2. Quantitative information of actual temperature can be inferred from Digital Readout, bits 22:16.

- **Thermal Threshold #2 Log (bit 9, R/WC0)** — Sticky bit that indicates whether the Thermal Threshold #2 has been reached since the last clearing of this bit or a reset. If bit 9 = 1, the Thermal Threshold #2 has been reached. Software may clear this bit by writing a zero.
- **Power Limitation Status (bit 10, RO)** — Indicates whether the processor is currently operating below OS-requested P-state (specified in IA32_PERF_CTL) or OS-requested clock modulation duty cycle (specified in IA32_CLOCK_MODULATION). This field is supported only if CPUID.06H:EAX[bit 4] = 1. Package level power limit notification can be delivered independently to IA32_PACKAGE_THERM_STATUS MSR.
- **Power Notification Log (bit 11, R/WCO)** — Sticky bit that indicates the processor went below OS-requested P-state or OS-requested clock modulation duty cycle since the last clearing of this or RESET. This field is supported only if CPUID.06H:EAX[bit 4] = 1. Package level power limit notification is indicated independently in IA32_PACKAGE_THERM_STATUS MSR.
- **Digital Readout (bits 22:16, RO)** — Digital temperature reading in 1 degree Celsius relative to the TCC activation temperature.

  0: TCC Activation temperature,

  1: (TCC Activation - 1) , etc. See the processor's data sheet for details regarding TCC activation.

  A lower reading in the Digital Readout field (bits 22:16) indicates a higher actual temperature.
- **Resolution in Degrees Celsius (bits 30:27, RO)** — Specifies the resolution (or tolerance) of the digital thermal sensor. The value is in degrees Celsius. It is recommended that new threshold values be offset from the current temperature by at least the resolution + 1 in order to avoid hysteresis of interrupt generation.
- **Reading Valid (bit 31, RO)** — Indicates if the digital readout in bits 22:16 is valid. The readout is valid if bit 31 = 1.

Changes to temperature can be detected using two thresholds (see Figure 14-32); one is set above and the other below the current temperature. These thresholds have the capability of generating interrupts using the core's local APIC which software must then service. Note that the local APIC entries used by these thresholds are also used by the Intel® Thermal Monitor; it is up to software to determine the source of a specific interrupt.

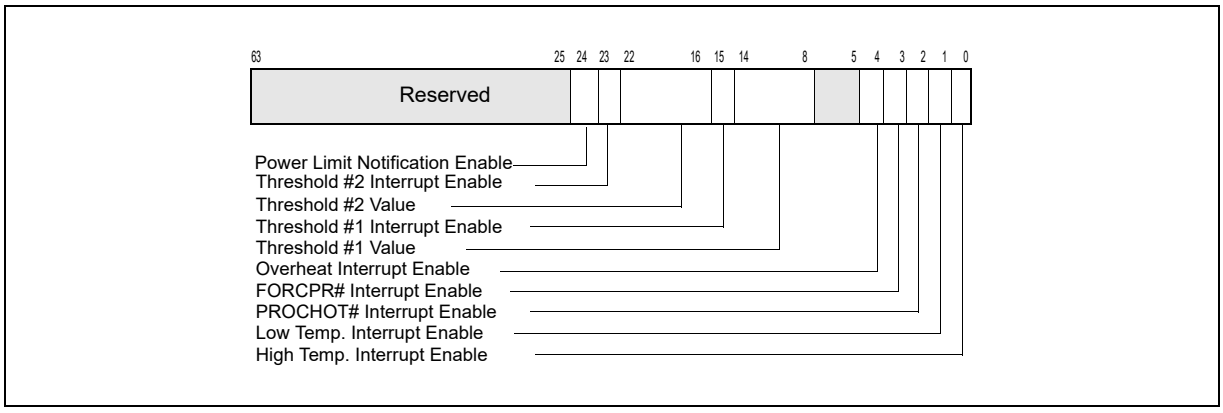


Figure 14-32. IA32_THERM_INTERRUPT Register

See Figure 14-32 for the layout of IA32_THERM_INTERRUPT MSR. Bit fields include:

- **High-Temperature Interrupt Enable (bit 0, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from low-temperature to a high-temperature threshold. Bit 0 = 0 (default) disables interrupts; bit 0 = 1 enables interrupts.
- **Low-Temperature Interrupt Enable (bit 1, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from high-temperature to a low-temperature (TCC de-activation). Bit 1 = 0 (default) disables interrupts; bit 1 = 1 enables interrupts.

- **PROCHOT# Interrupt Enable (bit 2, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when PROCHOT# has been asserted by another agent on the platform and the Bidirectional Prochot feature is enabled. Bit 2 = 0 disables the interrupt; bit 2 = 1 enables the interrupt.
- **FORCEPR# Interrupt Enable (bit 3, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when FORCEPR# has been asserted by another agent on the platform. Bit 3 = 0 disables the interrupt; bit 3 = 1 enables the interrupt.
- **Critical Temperature Interrupt Enable (bit 4, R/W)** — Enables the generation of an interrupt when the Critical Temperature Detector has detected a critical thermal condition. The recommended response to this condition is a system shutdown. Bit 4 = 0 disables the interrupt; bit 4 = 1 enables the interrupt.
- **Threshold #1 Value (bits 14:8, R/W)** — A temperature threshold, encoded relative to the TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Digital Readout and is used to generate the Thermal Threshold #1 Status and Log bits as well as the Threshold #1 thermal interrupt delivery.
- **Threshold #1 Interrupt Enable (bit 15, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Threshold #1 setting in any direction. Bit 15 = 1 enables the interrupt; bit 15 = 0 disables the interrupt.
- **Threshold #2 Value (bits 22:16, R/W)** —A temperature threshold, encoded relative to the TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Digital Readout and is used to generate the Thermal Threshold #2 Status and Log bits as well as the Threshold #2 thermal interrupt delivery.
- **Threshold #2 Interrupt Enable (bit 23, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Threshold #2 setting in any direction. Bit 23 = 1enables the interrupt; bit 23 = 0 disables the interrupt.
- **Power Limit Notification Enable (bit 24, R/W)** — Enables the generation of power notification events when the processor went below OS-requested P-state or OS-requested clock modulation duty cycle. This field is supported only if CPUID.06H:EAX[bit 4] = 1. Package level power limit notification can be enabled independently by IA32_PACKAGE_THERM_INTERRUPT MSR.

14.8.6 Power Limit Notification

Platform firmware may be capable of specifying a power limit to restrict power delivered to a platform component, such as a physical processor package. This constraint imposed by platform firmware may occasionally cause the processor to operate below OS-requested P or T-state. A power limit notification event can be delivered using the existing thermal LVT entry in the local APIC.

Software can enumerate the presence of the processor's support for power limit notification by verifying CPUID.06H:EAX[bit 4] = 1.

If CPUID.06H:EAX[bit 4] = 1, then IA32_THERM_INTERRUPT and IA32_THERM_STATUS provides the following facility to manage power limit notification:

- Bits 10 and 11 in IA32_THERM_STATUS informs software of the occurrence of processor operating below OS-requested P-state or clock modulation duty cycle setting (see Figure 14-31).
- Bit 24 in IA32_THERM_INTERRUPT enables the local APIC to deliver a thermal event when the processor went below OS-requested P-state or clock modulation duty cycle setting (see Figure 14-32).

14.9 PACKAGE LEVEL THERMAL MANAGEMENT

The thermal management facilities like IA32_THERM_INTERRUPT and IA32_THERM_STATUS are often implemented with a processor core granularity. To facilitate software manage thermal events from a package level granularity, two architectural MSR is provided for package level thermal management. The IA32_PACKAGE_THERM_STATUS and IA32_PACKAGE_THERM_INTERRUPT MSRs use similar interfaces as IA32_THERM_STATUS and IA32_THERM_INTERRUPT, but are shared in each physical processor package.

Software can enumerate the presence of the processor's support for package level thermal management facility (IA32_PACKAGE_THERM_STATUS and IA32_PACKAGE_THERM_INTERRUPT) by verifying CPUID.06H:EAX[bit 6] = 1.

The layout of IA32_PACKAGE_THERM_STATUS MSR is shown in Figure 14-33.

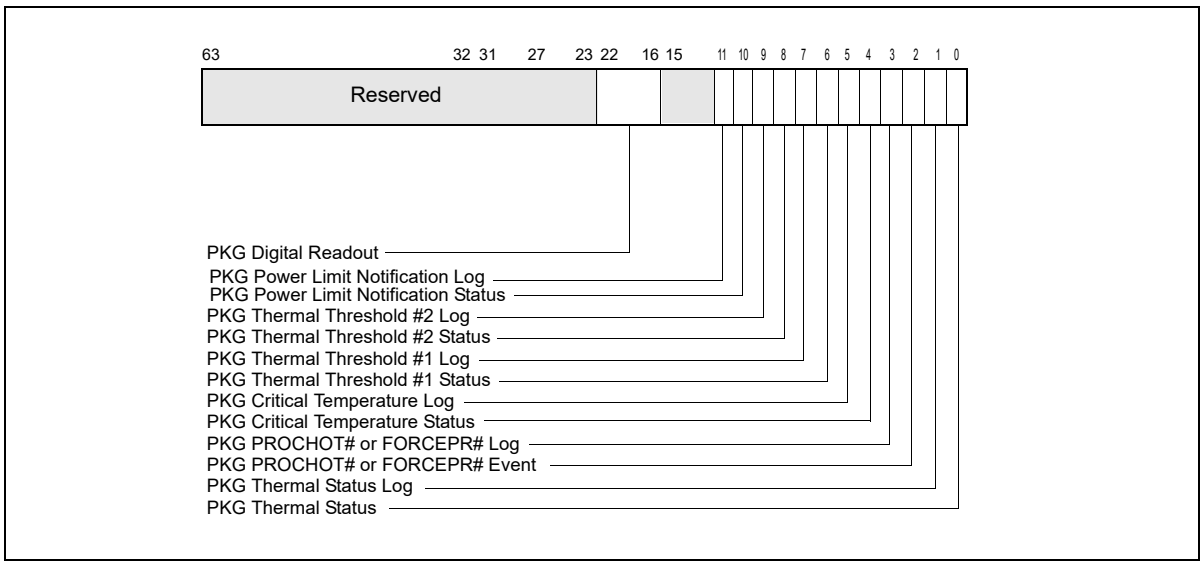


Figure 14-33. IA32_PACKAGE_THERM_STATUS Register

- **Package Thermal Status (bit 0, RO)** — This bit indicates whether the digital thermal sensor high-temperature output signal (PROCHOT#) for the package is currently active. Bit 0 = 1 indicates the feature is active. This bit may not be written by software; it reflects the state of the digital thermal sensor.
- **Package Thermal Status Log (bit 1, R/WC0)** — This is a sticky bit that indicates the history of the thermal sensor high temperature output signal (PROCHOT#) of the package. Bit 1 = 1 if package PROCHOT# has been asserted since a previous RESET or the last time software cleared the bit. Software may clear this bit by writing a zero.
- **Package PROCHOT# Event (bit 2, RO)** — Indicates whether package PROCHOT# is being asserted by another agent on the platform.
- **Package PROCHOT# Log (bit 3, R/WC0)** — Sticky bit that indicates whether package PROCHOT# has been asserted by another agent on the platform since the last clearing of this bit or a reset. If bit 3 = 1, package PROCHOT# has been externally asserted. Software may clear this bit by writing a zero.
- **Package Critical Temperature Status (bit 4, RO)** — Indicates whether the package critical temperature detector output signal is currently active. If bit 4 = 1, the package critical temperature detector output signal is currently active.
- **Package Critical Temperature Log (bit 5, R/WC0)** — Sticky bit that indicates whether the package critical temperature detector output signal has been asserted since the last clearing of this bit or reset. If bit 5 = 1, the output signal has been asserted. Software may clear this bit by writing a zero.
- **Package Thermal Threshold #1 Status (bit 6, RO)** — Indicates whether the actual package temperature is currently higher than or equal to the value set in Package Thermal Threshold #1. If bit 6 = 0, the actual temperature is lower. If bit 6 = 1, the actual temperature is greater than or equal to PTT#1. Quantitative information of actual package temperature can be inferred from Package Digital Readout, bits 22:16.
- **Package Thermal Threshold #1 Log (bit 7, R/WC0)** — Sticky bit that indicates whether the Package Thermal Threshold #1 has been reached since the last clearing of this bit or a reset. If bit 7 = 1, the Package Threshold #1 has been reached. Software may clear this bit by writing a zero.
- **Package Thermal Threshold #2 Status (bit 8, RO)** — Indicates whether actual package temperature is currently higher than or equal to the value set in Package Thermal Threshold #2. If bit 8 = 0, the actual

temperature is lower. If bit 8 = 1, the actual temperature is greater than or equal to PTT#2. Quantitative information of actual temperature can be inferred from Package Digital Readout, bits 22:16.

- **Package Thermal Threshold #2 Log (bit 9, R/WC0)** — Sticky bit that indicates whether the Package Thermal Threshold #2 has been reached since the last clearing of this bit or a reset. If bit 9 = 1, the Package Thermal Threshold #2 has been reached. Software may clear this bit by writing a zero.
- **Package Power Limitation Status (bit 10, RO)** — Indicates package power limit is forcing one ore more processors to operate below OS-requested P-state. Note that package power limit violation may be caused by processor cores or by devices residing in the uncore. Software can examine IA32_THERM_STATUS to determine if the cause originates from a processor core (see Figure 14-31).
- **Package Power Notification Log (bit 11, R/WCO)** — Sticky bit that indicates any processor in the package went below OS-requested P-state or OS-requested clock modulation duty cycle since the last clearing of this or RESET.
- **Package Digital Readout (bits 22:16, RO)** — Package digital temperature reading in 1 degree Celsius relative to the package TCC activation temperature.

  0: Package TCC Activation temperature,

  1: (PTCC Activation - 1) , etc. See the processor's data sheet for details regarding PTCC activation.

  A lower reading in the Package Digital Readout field (bits 22:16) indicates a higher actual temperature.

The layout of IA32_PACKAGE_THERM_INTERRUPT MSR is shown in Figure 14-34.

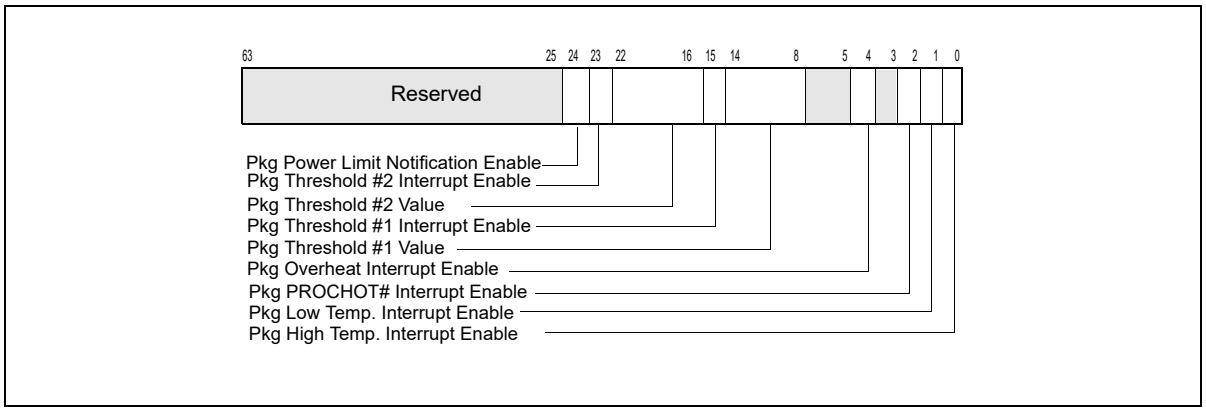


Figure 14-34. IA32_PACKAGE_THERM_INTERRUPT Register

- **Package High-Temperature Interrupt Enable (bit 0, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from low-temperature to a package high-temperature threshold. Bit 0 = 0 (default) disables interrupts; bit 0 = 1 enables interrupts.
- **Package Low-Temperature Interrupt Enable (bit 1, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from high-temperature to a low-temperature (TCC de-activation). Bit 1 = 0 (default) disables interrupts; bit 1 = 1 enables interrupts.
- **Package PROCHOT# Interrupt Enable (bit 2, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when Package PROCHOT# has been asserted by another agent on the platform and the Bidirectional Prochot feature is enabled. Bit 2 = 0 disables the interrupt; bit 2 = 1 enables the interrupt.
- **Package Critical Temperature Interrupt Enable (bit 4, R/W)** — Enables the generation of an interrupt when the Package Critical Temperature Detector has detected a critical thermal condition. The recommended response to this condition is a system shutdown. Bit 4 = 0 disables the interrupt; bit 4 = 1 enables the interrupt.
- **Package Threshold #1 Value (bits 14:8, R/W)** — A temperature threshold, encoded relative to the Package TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Package Digital Readout and is used to generate the Package Thermal Threshold #1 Status and Log bits as well as the Package Threshold #1 thermal interrupt delivery.

- **Package Threshold #1 Interrupt Enable (bit 15, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Package Threshold #1 setting in any direction. Bit 15 = 1 enables the interrupt; bit 15 = 0 disables the interrupt.
- **Package Threshold #2 Value (bits 22:16, R/W)** —A temperature threshold, encoded relative to the PTCC Activation temperature (using the same format as the Package Digital Readout). This threshold is compared against the Package Digital Readout and is used to generate the Package Thermal Threshold #2 Status and Log bits as well as the Package Threshold #2 thermal interrupt delivery.
- **Package Threshold #2 Interrupt Enable (bit 23, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Package Threshold #2 setting in any direction. Bit 23 = 1 enables the interrupt; bit 23 = 0 disables the interrupt.
- **Package Power Limit Notification Enable (bit 24, R/W)** — Enables the generation of package power notification events.

14.9.1 Support for Passive and Active cooling

Passive and active cooling may be controlled by the OS power management agent through ACPI control methods. On platforms providing package level thermal management facility described in the previous section, it is recommended that active cooling (FAN control) should be driven by measuring the package temperature using the IA32_PACKAGE_THERM_INTERRUPT MSR.

Passive cooling (frequency throttling) should be driven by measuring (a) the core and package temperatures, or (b) only the package temperature. If measured package temperature led the power management agent to choose which core to execute passive cooling, then all cores need to execute passive cooling. Core temperature is measured using the IA32_THERMAL_STATUS and IA32_THERMAL_INTERRUPT MSRs. The exact implementation details depend on the platform firmware and possible solutions include defining two different thermal zones (one for core temperature and passive cooling and the other for package temperature and active cooling).

14.10 PLATFORM SPECIFIC POWER MANAGEMENT SUPPORT

This section covers power management interfaces that are not architectural but addresses the power management needs of several platform specific components. Specifically, RAPL (Running Average Power Limit) interfaces provide mechanisms to enforce power consumption limit. Power limiting usages have specific usages in client and server platforms.

For client platform power limit control and for server platforms used in a data center, the following power and thermal related usages are desirable:

- Platform Thermal Management: Robust mechanisms to manage component, platform, and group-level thermals, either proactively or reactively (e.g., in response to a platform-level thermal trip point).
- Platform Power Limiting: More deterministic control over the system's power consumption, for example to meet battery life targets on rack-level or container-level power consumption goals within a datacenter.
- Power/Performance Budgeting: Efficient means to control the power consumed (and therefore the sustained performance delivered) within and across platforms.

The server and client usage models are addressed by RAPL interfaces, which expose multiple domains of power rationing within each processor socket. Generally, these RAPL domains may be viewed to include hierarchically:

- Package domain is the processor die.
- Memory domain includes the directly-attached DRAM; an additional power plane may constitute a separate domain.

In order to manage the power consumed across multiple sockets via RAPL, individual limits must be programmed for each processor complex. Programming specific RAPL domain across multiple sockets is not supported.

14.10.1 RAPL Interfaces

RAPL interfaces consist of non-architectural MSRs. Each RAPL domain supports the following set of capabilities, some of which are optional as stated below.

- Power limit - MSR interfaces to specify power limit, time window; lock bit, clamp bit etc.
- Energy Status - Power metering interface providing energy consumption information.
- Perf Status (Optional) - Interface providing information on the performance effects (regression) due to power limits. It is defined as a duration metric that measures the power limit effect in the respective domain. The meaning of duration is domain specific.
- Power Info (Optional) - Interface providing information on the range of parameters for a given domain, minimum power, maximum power etc.
- Policy (Optional) - 4-bit priority information that is a hint to hardware for dividing budget between sub-domains in a parent domain.

Each of the above capabilities requires specific units in order to describe them. Power is expressed in Watts, Time is expressed in Seconds, and Energy is expressed in Joules. Scaling factors are supplied to each unit to make the information presented meaningful in a finite number of bits. Units for power, energy, and time are exposed in the read-only MSR_RAPL_POWER_UNIT MSR.

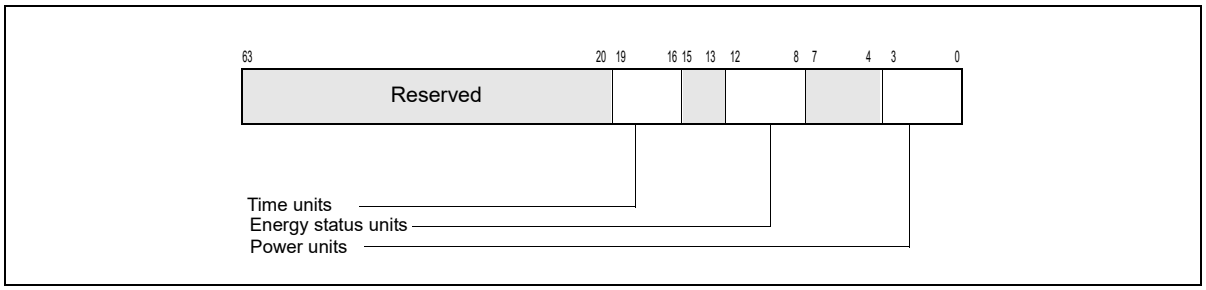


Figure 14-35. MSR_RAPL_POWER_UNIT Register

MSR_RAPL_POWER_UNIT (Figure 14-35) provides the following information across all RAPL domains:

- **Power Units** (bits 3:0): Power related information (in Watts) is based on the multiplier, 1/ 2^PU; where PU is an unsigned integer represented by bits 3:0. Default value is 0011b, indicating power unit is in 1/8 Watts increment.
- **Energy Status Units** (bits 12:8): Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 10000b, indicating energy status unit is in 15.3 micro-Joules increment.
- **Time Units** (bits 19:16): Time related information (in Seconds) is based on the multiplier, 1/ 2^TU; where TU is an unsigned integer represented by bits 19:16. Default value is 1010b, indicating time unit is in 976 micro-seconds increment.

14.10.2 RAPL Domains and Platform Specificity

The specific RAPL domains available in a platform vary across product segments. Platforms targeting the client segment support the following RAPL domain hierarchy:

- Package
- Two power planes: PP0 and PP1 (PP1 may reflect to uncore devices)

Platforms targeting the server segment support the following RAPL domain hierarchy:

- Package
- Power plane: PP0
- DRAM

Each level of the RAPL hierarchy provides a respective set of RAPL interface MSRs. Table 14-12 lists the RAPL MSR interfaces available for each RAPL domain. The power limit MSR of each RAPL domain is located at offset 0 relative to an MSR base address which is non-architectural (see Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*). The energy status MSR of each domain is located at offset 1 relative to the MSR base address of respective domain.

Table 14-12. RAPL MSR Interfaces and RAPL Domains

Domain	Power Limit (Offset 0)	Energy Status (Offset 1)	Policy (Offset 2)	Perf Status (Offset 3)	Power Info (Offset 4)
PKG	MSR_PKG_POWER_LIMIT	MSR_PKG_ENERGY_STATUS	RESERVED	MSR_PKG_PERF_STATUS	MSR_PKG_POWER_INFO
DRAM	MSR_DRAM_POWER_LIMIT	MSR_DRAM_ENERGY_STATUS	RESERVED	MSR_DRAM_PERF_STATUS	MSR_DRAM_POWER_INFO
PP0	MSR_PP0_POWER_LIMIT	MSR_PP0_ENERGY_STATUS	MSR_PP0_POLICY	MSR_PP0_PERF_STATUS	RESERVED
PP1	MSR_PP1_POWER_LIMIT	MSR_PP1_ENERGY_STATUS	MSR_PP1_POLICY	RESERVED	RESERVED

The presence of the optional MSR interfaces (the three right-most columns of Table 14-12) may be model-specific. See Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4* for details.

14.10.3 Package RAPL Domain

The MSR interfaces defined for the package RAPL domain are:

- MSR_PKG_POWER_LIMIT allows software to set power limits for the package and measurement attributes associated with each limit,
- MSR_PKG_ENERGY_STATUS reports measured actual energy usage,
- MSR_PKG_POWER_INFO reports the package power range information for RAPL usage.

MSR_PKG_PERF_STATUS can report the performance impact of power limiting, but its availability may be model-specific.

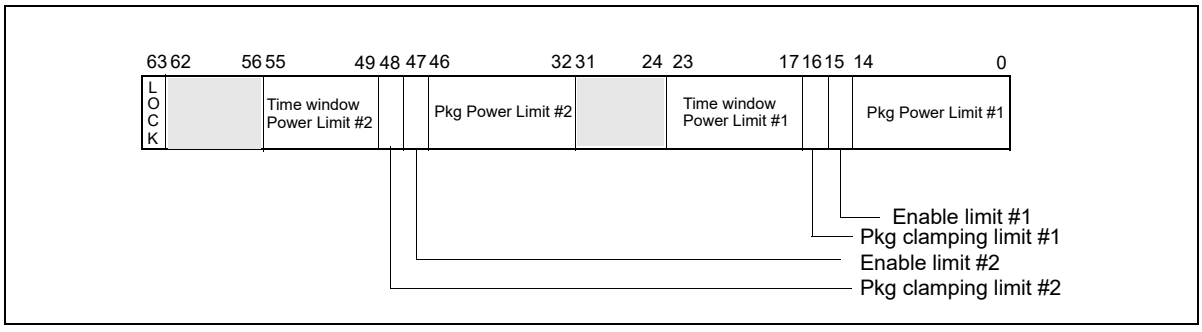


Figure 14-36. MSR_PKG_POWER_LIMIT Register

MSR_PKG_POWER_LIMIT allows a software agent to define power limitation for the package domain. Power limitation is defined in terms of average power usage (Watts) over a time window specified in MSR_PKG_POWER_LIMIT. Two power limits can be specified, corresponding to time windows of different sizes. Each power limit provides independent clamping control that would permit the processor cores to go below OS-requested state to meet the power

limits. A lock mechanism allow the software agent to enforce power limit settings. Once the lock bit is set, the power limit settings are static and un-modifiable until next RESET.

The bit fields of MSR_PKG_POWER_LIMIT (Figure 14-36) are:

- **Package Power Limit #1**(bits 14:0): Sets the average power usage limit of the package domain corresponding to time window # 1. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #1**(bit 15): 0 = disabled; 1 = enabled.
- **Package Clamping Limitation #1** (bit 16): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit #1** (bits 23:17): Indicates the time window for power limit #1

  Time limit = 2^Y * (1.0 + Z/4.0) * Time_Unit

  Here "Y" is the unsigned integer value represented. by bits 21:17, "Z" is an unsigned integer represented by bits 23:22. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Package Power Limit #2**(bits 46:32): Sets the average power usage limit of the package domain corresponding to time window # 2. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #2**(bit 47): 0 = disabled; 1 = enabled.
- **Package Clamping Limitation #2** (bit 48): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit #2** (bits 55:49): Indicates the time window for power limit #2

  Time limit = 2^Y * (1.0 + Z/4.0) * Time_Unit

  Here "Y" is the unsigned integer value represented. by bits 53:49, "Z" is an unsigned integer represented by bits 55:54. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT. This field may have a hard-coded value in hardware and ignores values written by software.
- **Lock** (bit 63): If set, all write attempts to this MSR are ignored until next RESET.

MSR_PKG_ENERGY_STATUS is a read-only MSR. It reports the actual energy use for the package domain. This MSR is updated every ~1msec. It has a wraparound time of around 60 secs when power consumption is high, and may be longer otherwise.

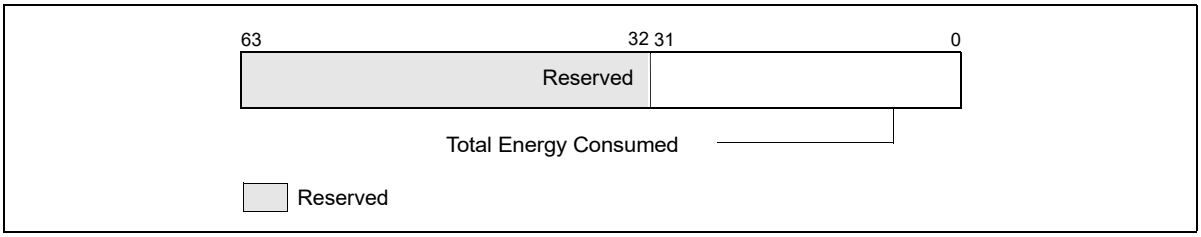


Figure 14-37. MSR_PKG_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since that last time this register is cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_PKG_POWER_INFO is a read-only MSR. It reports the package power range information for RAPL usage. This MSR provides maximum/minimum values (derived from electrical specification), thermal specification power of the package domain. It also provides the largest possible time window for software to program the RAPL interface.

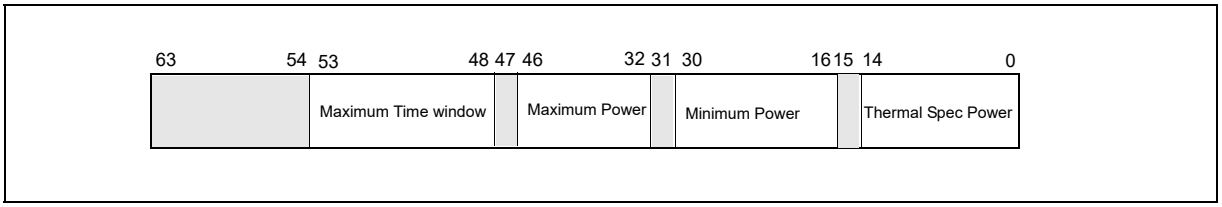


Figure 14-38. MSR_PKG_POWER_INFO Register

- **Thermal Spec Power** (bits 14:0): The unsigned integer value is the equivalent of thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Minimum Power** (bits 30:16): The unsigned integer value is the equivalent of minimum power derived from electrical spec of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Power** (bits 46:32): The unsigned integer value is the equivalent of maximum power derived from the electrical spec of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Time Window** (bits 53:48): The unsigned integer value is the equivalent of largest acceptable value to program the time window of MSR_PKG_POWER_LIMIT. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

MSR_PKG_PERF_STATUS is a read-only MSR. It reports the total time for which the package was throttled due to the RAPL power limits. Throttling in this context is defined as going below the OS-requested P-state or T-state. It has a wrap-around time of many hours. The availability of this MSR is platform specific (see Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*).

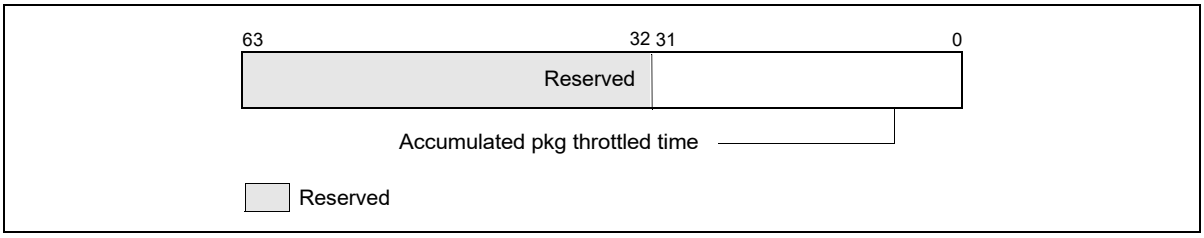


Figure 14-39. MSR_PKG_PERF_STATUS MSR

- **Accumulated Package Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the package has throttled. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

14.10.4 PP0/PP1 RAPL Domains

The MSR interfaces defined for the PP0 and PP1 domains are identical in layout. Generally, PP0 refers to the processor cores. The availability of PP1 RAPL domain interface is platform-specific. For a client platform, the PP1 domain refers to the power plane of a specific device in the uncore. For server platforms, the PP1 domain is not supported, but its PP0 domain supports the MSR_PP0_PERF_STATUS interface.

- MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT allow software to set power limits for the respective power plane domain.
- MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS report actual energy usage on a power plane.

- MSR_PP0_POLICY/MSR_PP1_POLICY allow software to adjust balance for respective power plane.

MSR_PP0_PERF_STATUS can report the performance impact of power limiting, but it is not available in client platforms.

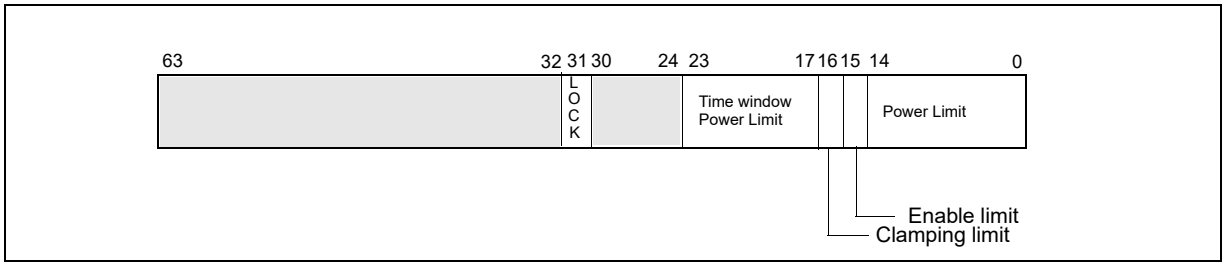


Figure 14-40. MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT Register

MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT allow a software agent to define power limitation for the respective power plane domain. A lock mechanism in each power plane domain allows the software agent to enforce power limit settings independently. Once a lock bit is set, the power limit settings in that power plane are static and un-modifiable until next RESET.

The bit fields of MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT (Figure 14-40) are:

- **Power Limit** (bits 14:0): Sets the average power usage limit of the respective power plane domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit** (bit 15): 0 = disabled; 1 = enabled.
- **Clamping Limitation** (bit 16): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit** (bits 23:17): Indicates the length of time window over which the power limit #1 will be used by the processor. The numeric value encoded by bits 23:17 is represented by the product of 2^Y *F; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 23:22, Y is an unsigned integer represented by bits 21:17. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Lock** (bit 31): If set, all write attempts to the MSR and corresponding policy MSR_PP0_POLICY/MSR_PP1_POLICY are ignored until next RESET.

MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS are read-only MSRs. They report the actual energy use for the respective power plane domains. These MSRs are updated every ~1msec.

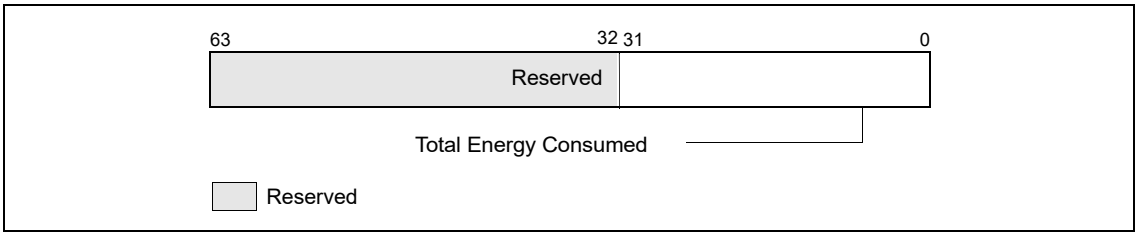


Figure 14-41. MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since the last time this register was cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_PP0_POLICY/MSR_PP1_POLICY provide balance power policy control for each power plane by providing inputs to the power budgeting management algorithm. On platforms that support PP0 (IA cores) and PP1 (uncore

graphic device), the default values give priority to the non-IA power plane. These MSRs enable the PCU to balance power consumption between the IA cores and uncore graphic device.

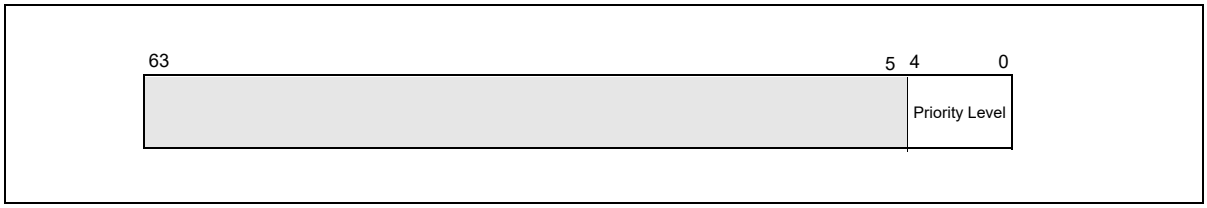


Figure 14-42. MSR_PP0_POLICY/MSR_PP1_POLICY Register

- **Priority Level** (bits 4:0): Priority level input to the PCU for respective power plane. PP0 covers the IA processor cores, PP1 covers the uncore graphic device. The value 31 is considered highest priority.

MSR_PP0_PERF_STATUS is a read-only MSR. It reports the total time for which the PP0 domain was throttled due to the power limits. This MSR is supported only in server platform. Throttling in this context is defined as going below the OS-requested P-state or T-state.

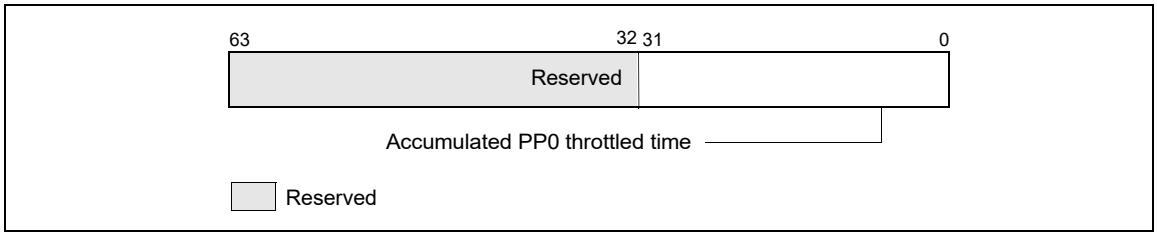


Figure 14-43. MSR_PP0_PERF_STATUS MSR

- **Accumulated PP0 Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the PP0 domain has throttled. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

14.10.5 DRAM RAPL Domain

The MSR interfaces defined for the DRAM domains are supported only in the server platform. The MSR interfaces are:

- MSR_DRAM_POWER_LIMIT allows software to set power limits for the DRAM domain and measurement attributes associated with each limit.
- MSR_DRAM_ENERGY_STATUS reports measured actual energy usage.
- MSR_DRAM_POWER_INFO reports the DRAM domain power range information for RAPL usage.
- MSR_DRAM_PERF_STATUS can report the performance impact of power limiting.

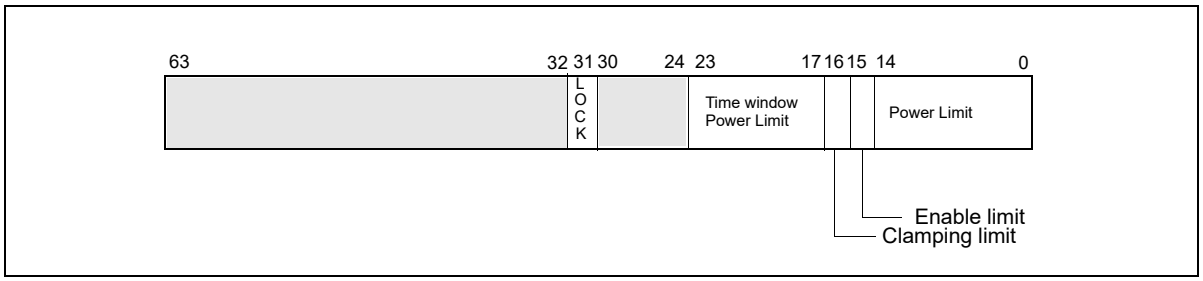


Figure 14-44. MSR_DRAM_POWER_LIMIT Register

MSR_DRAM_POWER_LIMIT allows a software agent to define power limitation for the DRAM domain. Power limitation is defined in terms of average power usage (Watts) over a time window specified in MSR_DRAM_POWER_LIMIT. A power limit can be specified along with a time window. A lock mechanism allow the software agent to enforce power limit settings. Once the lock bit is set, the power limit settings are static and unmodifiable until next RESET.

The bit fields of MSR_DRAM_POWER_LIMIT (Figure 14-44) are:

- **DRAM Power Limit #1**(bits 14:0): Sets the average power usage limit of the DRAM domain corresponding to time window # 1. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #1**(bit 15): 0 = disabled; 1 = enabled.
- **Time Window for Power Limit** (bits 23:17): Indicates the length of time window over which the power limit will be used by the processor. The numeric value encoded by bits 23:17 is represented by the product of 2^Y *F; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 23:22, Y is an unsigned integer represented by bits 21:17. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Lock** (bit 31): If set, all write attempts to this MSR are ignored until next RESET.

MSR_DRAM_ENERGY_STATUS is a read-only MSR. It reports the actual energy use for the DRAM domain. This MSR is updated every ~1msec.

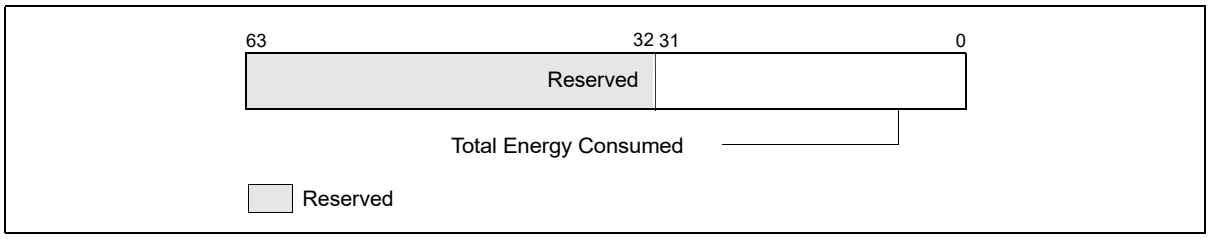


Figure 14-45. MSR_DRAM_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since that last time this register is cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_DRAM_POWER_INFO is a read-only MSR. It reports the DRAM power range information for RAPL usage. This MSR provides maximum/minimum values (derived from electrical specification), thermal specification power of the DRAM domain. It also provides the largest possible time window for software to program the RAPL interface.

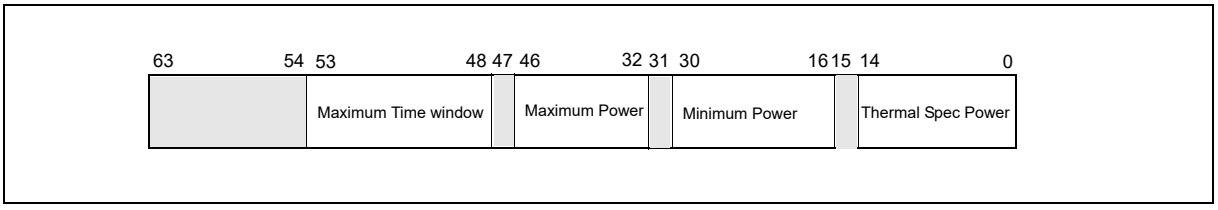


Figure 14-46. MSR_DRAM_POWER_INFO Register

- **Thermal Spec Power** (bits 14:0): The unsigned integer value is the equivalent of thermal specification power of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Minimum Power** (bits 30:16): The unsigned integer value is the equivalent of minimum power derived from electrical spec of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Power** (bits 46:32): The unsigned integer value is the equivalent of maximum power derived from the electrical spec of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.

- **Maximum Time Window** (bits 53:48): The unsigned integer value is the equivalent of largest acceptable value to program the time window of MSR_DRAM_POWER_LIMIT. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

MSR_DRAM_PERF_STATUS is a read-only MSR. It reports the total time for which the package was throttled due to the RAPL power limits. Throttling in this context is defined as going below the OS-requested P-state or T-state. It has a wrap-around time of many hours. The availability of this MSR is platform specific (see Chapter 2, "Model-Specific Registers (MSRs)" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4*).

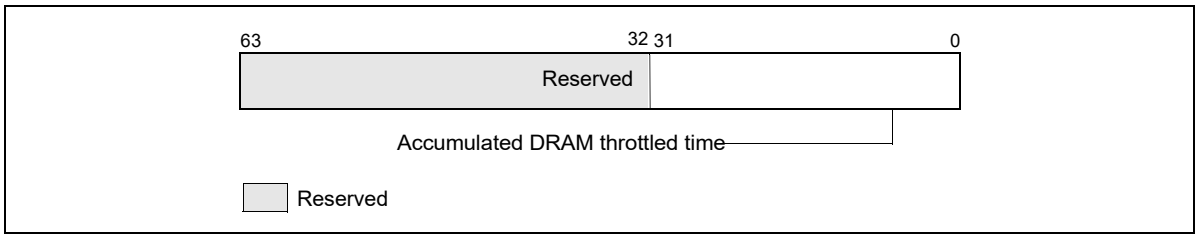


Figure 14-47. MSR_DRAM_PERF_STATUS MSR

- **Accumulated Package Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the DRAM domain has throttled. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

16. Updates to Chapter 16, Volume 3B

Change bars and green text show changes to Chapter 16 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

Changes to this chapter: Update to Table 16-2, "Incremental Decoding Information: Processor Family 06H Machine Error Codes For Machine Check".

CHAPTER 16
INTERPRETING MACHINE-CHECK ERROR CODES

Encoding of the model-specific and other information fields is different across processor families. The differences are documented in the following sections.

16.1 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY 06H MACHINE ERROR CODES FOR MACHINE CHECK

Section 16.1 provides information for interpreting additional model-specific fields for external bus errors relating to processor family 06H. The references to processor family 06H refers to only IA-32 processors with CPUID signatures listed in Table 16-1.

Table 16-1. CPUID DisplayFamily_DisplayModel Signatures for Processor Family 06H

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_0EH	Intel Core Duo, Intel Core Solo processors
06_0DH	Intel Pentium M processor
06_09H	Intel Pentium M processor
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

These errors are reported in the IA32_MCi_STATUS MSRs. They are reported architecturally as compound errors with a general form of *0000 1PPT RRRR IILL* in the MCA error code field. See Chapter 15 for information on the interpretation of compound error codes. Incremental decoding information is listed in Table 16-2.

Table 16-2. Incremental Decoding Information: Processor Family 06H Machine Error Codes For Machine Check

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0		
Model specific errors	18:16	Reserved	Reserved
Model specific errors	24:19	Bus queue request type	000000 for BQ_DCU_READ_TYPE error 000010 for BQ_IFU_DEMAND_TYPE error 000011 for BQ_IFU_DEMAND_NC_TYPE error 000100 for BQ_DCU_RFO_TYPE error 000101 for BQ_DCU_RFO_LOCK_TYPE error 000110 for BQ_DCU_ITOM_TYPE error 001000 for BQ_DCU_WB_TYPE error 001010 for BQ_DCU_WCEVICT_TYPE error 001011 for BQ_DCU_WCLINE_TYPE error 001100 for BQ_DCU_BTM_TYPE error

Table 16-2. Incremental Decoding Information: Processor Family 06H Machine Error Codes For Machine Check

Type	Bit No.	Bit Function	Bit Description
			001101 for BQ_DCU_INTACK_TYPE error 001110 for BQ_DCU_INVALL2_TYPE error 001111 for BQ_DCU_FLUSHL2_TYPE error 010000 for BQ_DCU_PART_RD_TYPE error 010010 for BQ_DCU_PART_WR_TYPE error 010100 for BQ_DCU_SPEC_CYC_TYPE error 011000 for BQ_DCU_IO_RD_TYPE error 011001 for BQ_DCU_IO_WR_TYPE error 011100 for BQ_DCU_LOCK_RD_TYPE error 011110 for BQ_DCU_SPLOCK_RD_TYPE error 011101 for BQ_DCU_LOCK_WR_TYPE error
Model specific errors	27:25	Bus queue error type	000 for BQ_ERR_HARD_TYPE error 001 for BQ_ERR_DOUBLE_TYPE error 010 for BQ_ERR_AERR2_TYPE error 100 for BQ_ERR_SINGLE_TYPE error 101 for BQ_ERR_AERR1_TYPE error
Model specific errors	28	FRC error	1 if FRC error active
	29	BERR	1 if BERR is driven
	30	Internal BINIT	1 if BINIT driven for this processor
	31	Reserved	Reserved
Other information	34:32	Reserved	Reserved
	35	External BINIT	1 if BINIT is received from external bus.
	36	Response parity error	This bit is asserted in IA32_MC*i*_STATUS if this component has received a parity error on the RS[2:0]# pins for a response transaction. The RS signals are checked by the RSP# external pin.
	37	Bus BINIT	This bit is asserted in IA32_MC*i*_STATUS if this component has received a hard error response on a split transaction one access that has needed to be split across the 64-bit external bus interface into two accesses).
	38	Timeout BINIT	This bit is asserted in IA32_MC*i*_STATUS if this component has experienced a ROB time-out, which indicates that no micro-instruction has been retired for a predetermined period of time. A ROB time-out occurs when the 15-bit ROB time-out counter carries a 1 out of its high order bit. [2] The timer is cleared when a micro-instruction retires, an exception is detected by the core processor, RESET is asserted, or when a ROB BINIT occurs. The ROB time-out counter is prescaled by the 8-bit PIC timer which is a divide by 128 of the bus clock (the bus clock is 1:2, 1:3, 1:4 of the core clock[3]). When a carry out of the 8-bit PIC timer occurs, the ROB counter counts up by one. While this bit is asserted, it cannot be overwritten by another error.
	41:39	Reserved	Reserved
	42	Hard error	This bit is asserted in IA32_MC*i*_STATUS if this component has initiated a bus transactions which has received a hard error response. While this bit is asserted, it cannot be overwritten.

Table 16-2. Incremental Decoding Information: Processor Family 06H Machine Error Codes For Machine Check

Type	Bit No.	Bit Function	Bit Description
	43	IERR	This bit is asserted in IA32_MC*i*_STATUS if this component has experienced a failure that causes the IERR pin to be asserted. While this bit is asserted, it cannot be overwritten.
	44	AERR	This bit is asserted in IA32_MC*i*_STATUS if this component has initiated 2 failing bus transactions which have failed due to Address Parity Errors AERR asserted). While this bit is asserted, it cannot be overwritten.
	45	UECC	The Uncorrectable ECC error bit is asserted in IA32_MC*i*_STATUS for uncorrected ECC errors. While this bit is asserted, the ECC syndrome field will not be overwritten.
	46	CECC	The correctable ECC error bit is asserted in IA32_MC*i*_STATUS for corrected ECC errors.
	54:47	ECC syndrome	The ECC syndrome field in IA32_MCi_STATUS contains the 8-bit ECC syndrome only if the error was a correctable/uncorrectable ECC error and there wasn't a previous valid ECC error syndrome logged in IA32_MCi_STATUS. A previous valid ECC error in IA32_MCi_STATUS is indicated by IA32_MCi_STATUS.bit45 uncorrectable error occurred) being asserted. After processing an ECC error, machine-check handling software should clear IA32_MCi_STATUS.bit45 so that future ECC error syndromes can be logged.
	56:55	Reserved	Reserved.
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.
2. For processors with a CPUID signature of 06_0EH, a ROB time-out occurs when the 23-bit ROB time-out counter carries a 1 out of its high order bit.
3. For processors with a CPUID signature of 6_06_60H and later, the PIC timer will count crystal clock cycles.

16.2 INCREMENTAL DECODING INFORMATION: INTEL CORE 2 PROCESSOR FAMILY MACHINE ERROR CODES FOR MACHINE CHECK

Table 16-4 provides information for interpreting additional model-specific fields for external bus errors relating to processor based on Intel Core microarchitecture, which implements the P4 bus specification. Table 16-3 lists the CPUID signatures for Intel 64 processors that are covered by Table 16-4. These errors are reported in the IA32_MCi_STATUS MSRs. They are reported architecturally as compound errors with a general form of *0000 1PPT RRRR IILL* in the MCA error code field. See Chapter 15 for information on the interpretation of compound error codes.

Table 16-3. CPUID DisplayFamily_DisplayModel Signatures for Processors Based on Intel Core Microarchitecture

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_1DH	Intel Xeon Processor 7400 series.
06_17H	Intel Xeon Processor 5200, 5400 series, Intel Core 2 Quad processor Q9650.
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.

Table 16-4. Incremental Bus Error Codes of Machine Check for Processors Based on Intel Core Microarchitecture

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0		
Model specific errors	18:16	Reserved	Reserved
Model specific errors	24:19	Bus queue request type	'000001 for BQ_PREF_READ_TYPE error 000000 for BQ_DCU_READ_TYPE error 000010 for BQ_IFU_DEMAND_TYPE error 000011 for BQ_IFU_DEMAND_NC_TYPE error 000100 for BQ_DCU_RFO_TYPE error 000101 for BQ_DCU_RFO_LOCK_TYPE error 000110 for BQ_DCU_ITOM_TYPE error 001000 for BQ_DCU_WB_TYPE error 001010 for BQ_DCU_WCEVICT_TYPE error 001011 for BQ_DCU_WCLINE_TYPE error 001100 for BQ_DCU_BTM_TYPE error 001101 for BQ_DCU_INTACK_TYPE error 001110 for BQ_DCU_INVALL2_TYPE error 001111 for BQ_DCU_FLUSHL2_TYPE error 010000 for BQ_DCU_PART_RD_TYPE error 010010 for BQ_DCU_PART_WR_TYPE error 010100 for BQ_DCU_SPEC_CYC_TYPE error 011000 for BQ_DCU_IO_RD_TYPE error 011001 for BQ_DCU_IO_WR_TYPE error 011100 for BQ_DCU_LOCK_RD_TYPE error 011110 for BQ_DCU_SPLOCK_RD_TYPE error 011101 for BQ_DCU_LOCK_WR_TYPE error 100100 for BQ_L2_WI_RFO_TYPE error 100110 for BQ_L2_WI_ITOM_TYPE error
Model specific errors	27:25	Bus queue error type	'001 for Address Parity Error '010 for Response Hard Error '011 for Response Parity Error
Model specific errors	28	MCE Driven	1 if MCE is driven
	29	MCE Observed	1 if MCE is observed
	30	Internal BINIT	1 if BINIT driven for this processor
	31	BINIT Observed	1 if BINIT is observed for this processor
Other information	33:32	Reserved	Reserved
	34	PIC and FSB data parity	Data Parity detected on either PIC or FSB access
	35	Reserved	Reserved

Table 16-4. Incremental Bus Error Codes of Machine Check for Processors Based on Intel Core Microarchitecture (Contd.)

Type	Bit No.	Bit Function	Bit Description
	36	Response parity error	This bit is asserted in IA32_MC*i*_STATUS if this component has received a parity error on the RS[2:0]# pins for a response transaction. The RS signals are checked by the RSP# external pin.
	37	FSB address parity	Address parity error detected: 1 = Address parity error detected 0 = No address parity error
	38	Timeout BINIT	This bit is asserted in IA32_MC*i*_STATUS if this component has experienced a ROB time-out, which indicates that no micro-instruction has been retired for a predetermined period of time. A ROB time-out occurs when the 23-bit ROB time-out counter carries a 1 out of its high order bit. The timer is cleared when a micro-instruction retires, an exception is detected by the core processor, RESET is asserted, or when a ROB BINIT occurs. The ROB time-out counter is prescaled by the 8-bit PIC timer which is a divide by 128 of the bus clock the bus clock is 1:2, 1:3, 1:4 of the core clock). When a carry out of the 8-bit PIC timer occurs, the ROB counter counts up by one. While this bit is asserted, it cannot be overwritten by another error.
	41:39	Reserved	Reserved
	42	Hard error	This bit is asserted in IA32_MC*i*_STATUS if this component has initiated a bus transactions which has received a hard error response. While this bit is asserted, it cannot be overwritten.
	43	IERR	This bit is asserted in IA32_MC*i*_STATUS if this component has experienced a failure that causes the IERR pin to be asserted. While this bit is asserted, it cannot be overwritten.
	44	Reserved	Reserved
	45	Reserved	Reserved
	46	Reserved	Reserved
	54:47	Reserved	Reserved
	56:55	Reserved	Reserved.
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.2.1 Model-Specific Machine Check Error Codes for Intel Xeon Processor 7400 Series

Intel Xeon processor 7400 series has machine check register banks that generally follows the description of Chapter 15 and Section 16.2. Additional error codes specific to Intel Xeon processor 7400 series is describe in this section.

MC4_STATUS[63:0] is the main error logging for the processor's L3 and front side bus errors for Intel Xeon processor 7400 series. It supports the L3 Errors, Bus and Interconnect Errors Compound Error Codes in the MCA Error Code Field.

16.2.1.1 Processor Machine Check Status Register Incremental MCA Error Code Definition

Intel Xeon processor 7400 series use compound MCA Error Codes for logging its Bus internal machine check errors, L3 Errors, and Bus/Interconnect Errors. It defines incremental Machine Check error types (IA32_MC6_STATUS[15:0]) beyond those defined in Chapter 15. Table 16-5 lists these incremental MCA error code types that apply to IA32_MC6_STATUS. Error code details are specified in MC6_STATUS [31:16] (see Section 16.2.2), the "Model Specific Error Code" field. The information in the "Other_Info" field (MC4_STATUS[56:32]) is common to the three processor error types and contains a correctable event count and specifies the MC6_MISC register format.

Table 16-5. Incremental MCA Error Code Types for Intel Xeon Processor 7400

Processor MCA_Error_Code (MC6_STATUS[15:0])			
Type	**Error Code**	**Binary Encoding**	**Meaning**
C	Internal Error	0000 0100 0000 0000	Internal Error Type Code
B	Bus and Interconnect Error	0000 100x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations
		0000 101x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations
		0000 110x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations
		0000 1110 0000 1111	Bus and Interconnection Error Type Code
		0000 1111 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations

The **Bold faced** binary encodings are the only encodings used by the processor for MC4_STATUS[15:0].

16.2.2 Intel Xeon Processor 7400 Model Specific Error Code Field

16.2.2.1 Processor Model Specific Error Code Field Type B: Bus and Interconnect Error

Note: The Model Specific Error Code field in MC6_STATUS (bits 31:16).

Table 16-6. Type B Bus and Interconnect Error Codes

Bit Num	Sub-Field Name	Description
16	FSB Request Parity	Parity error detected during FSB request phase
19:17		Reserved
20	FSB Hard Fail Response	"Hard Failure" response received for a local transaction
21	FSB Response Parity	Parity error on FSB response field detected
22	FSB Data Parity	FSB data parity error on inbound data detected
31:23	---	Reserved

16.2.2.2 Processor Model Specific Error Code Field Type C: Cache Bus Controller Error

Table 16-7. Type C Cache Bus Controller Error Codes

MC4_STATUS[31:16] (MSCE) Value	Error Description
0000_0000_0000_0001 0001H	Inclusion Error from Core 0
0000_0000_0000_0010 0002H	Inclusion Error from Core 1
0000_0000_0000_0011 0003H	Write Exclusive Error from Core 0
0000_0000_0000_0100 0004H	Write Exclusive Error from Core 1
0000_0000_0000_0101 0005H	Inclusion Error from FSB
0000_0000_0000_0110 0006H	SNP Stall Error from FSB
0000_0000_0000_0111 0007H	Write Stall Error from FSB
0000_0000_0000_1000 0008H	FSB Arb Timeout Error
0000_0000_0000_1010 000AH	Inclusion Error from Core 2
0000_0000_0000_1011 000BH	Write Exclusive Error from Core 2
0000_0010_0000_0000 0200H	Internal Timeout error
0000_0011_0000_0000 0300H	Internal Timeout Error
0000_0100_0000_0000 0400H	Intel® Cache Safe Technology Queue Full Error or Disabled-ways-in-a-set overflow
0000_0101_0000_0000 0500H	Quiet cycle Timeout Error (correctable)
1100_0000_0000_0010 C002H	Correctable ECC event on outgoing Core 0 data
1100_0000_0000_0100 C004H	Correctable ECC event on outgoing Core 1 data
1100_0000_0000_1000 C008H	Correctable ECC event on outgoing Core 2 data
1110_0000_0000_0010 E002H	Uncorrectable ECC error on outgoing Core 0 data
1110_0000_0000_0100 E004H	Uncorrectable ECC error on outgoing Core 1 data
1110_0000_0000_1000 E008H	Uncorrectable ECC error on outgoing Core 2 data
— all other encodings —	Reserved

16.3 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR 3400, 3500, 5500 SERIES, MACHINE ERROR CODES FOR MACHINE CHECK

Table 16-8 through Table 16-12 provide information for interpreting additional model-specific fields for memory controller errors relating to the Intel® Xeon® processor 3400, 3500, 5500 series with CPUID DisplayFamily_DisplaySignature 06_1AH, which supports Intel QuickPath Interconnect links. Incremental MC error codes related to the Intel QPI links are reported in the register banks IA32_MC0 and IA32_MC1, incremental error codes for internal machine check is reported in the register bank IA32_MC7, and incremental error codes for the memory controller unit is reported in the register banks IA32_MC8.

16.3.1 Intel QPI Machine Check Errors

Table 16-8. Intel QPI Machine Check Error Codes for IA32_MC0_STATUS and IA32_MC1_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Bus error format: 1PPTRRRRIILL
Model specific errors			
	16	Header Parity	if 1, QPI Header had bad parity
	17	Data Parity	If 1, QPI Data packet had bad parity
	18	Retries Exceeded	If 1, number of QPI retries was exceeded
	19	Received Poison	if 1, Received a data packet that was marked as poisoned by the sender
	21:20	Reserved	Reserved
	22	Unsupported Message	If 1, QPI received a message encoding it does not support
	23	Unsupported Credit	If 1, QPI credit type is not supported.
	24	Receive Flit Overrun	If 1, Sender sent too many QPI flits to the receiver.
	25	Received Failed Response	If 1, Indicates that sender sent a failed response to receiver.
	26	Receiver Clock Jitter	If 1, clock jitter detected in the internal QPI clocking
	56:27	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

Table 16-9. Intel QPI Machine Check Error Codes for IA32_MC0_MISC and IA32_MC1_MISC

Type	Bit No.	Bit Function	Bit Description
Model specific errors[1]			
	7:0	QPI Opcode	Message class and opcode from the packet with the error
	13:8	RTId	QPI Request Transaction ID
	15:14	Reserved	Reserved
	18:16	RHNID	QPI Requestor/Home Node ID
	23:19	Reserved	Reserved
	24	IIB	QPI Interleave/Head Indication Bit

NOTES:

1. Which of these fields are valid depends on the error type.

16.3.2 Internal Machine Check Errors

Table 16-10. Machine Check Error Codes for IA32_MC7_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	
Model specific errors			

Type	Bit No.	Bit Function	Bit Description
	23:16	Reserved	Reserved
	31:24	Reserved except for the following	00h - No Error 03h - Reset firmware did not complete 08h - Received an invalid CMPD 0Ah - Invalid Power Management Request 0Dh - Invalid S-state transition 11h - VID controller does not match POC controller selected 1Ah - MSID from POC does not match CPU MSID
	56:32	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.3.3 Memory Controller Errors

Table 16-11. Incremental Memory Controller Error Codes of Machine Check for IA32_MC8_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Memory error format: 1MMMCCCC
Model specific errors			
	16	Read ECC error	if 1, ECC occurred on a read
	17	RAS ECC error	If 1, ECC occurred on a scrub
	18	Write parity error	If 1, bad parity on a write
	19	Redundancy loss	if 1, Error in half of redundant memory
	20	Reserved	Reserved
	21	Memory range error	If 1, Memory access out of range
	22	RTID out of range	If 1, Internal ID invalid
	23	Address parity error	If 1, bad address parity
	24	Byte enable parity error	If 1, bad enable parity
Other information	37:25	Reserved	Reserved
	52:38	CORE_ERR_CNT	Corrected error count
	56:53	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

Table 16-12. Incremental Memory Controller Error Codes of Machine Check for IA32_MC8_MISC

Type	Bit No.	Bit Function	Bit Description
Model specific errors[1]			
	7:0	RTId	Transaction Tracker ID
	15:8	Reserved	Reserved
	17:16	DIMM	DIMM ID which got the error
	19:18	Channel	Channel ID which got the error
	31:20	Reserved	Reserved
	63:32	Syndrome	ECC Syndrome

NOTES:
1. Which of these fields are valid depends on the error type.

16.4 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR E5 FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

Table 16-13 through Table 16-15 provide information for interpreting additional model-specific fields for memory controller errors relating to the Intel® Xeon® processor E5 Family with CPUID DisplayFamily_DisplaySignature 06_2DH, which supports Intel QuickPath Interconnect links. Incremental MC error codes related to the Intel QPI links are reported in the register banks IA32_MC6 and IA32_MC7, incremental error codes for internal machine check error from PCU controller is reported in the register bank IA32_MC4, and incremental error codes for the memory controller unit is reported in the register banks IA32_MC8-IA32_MC11.

16.4.1 Internal Machine Check Errors

Table 16-13. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	
Model specific errors	19:16	Reserved except for the following	0000b - No Error 0001b - Non_IMem_Sel 0010b - I_Parity_Error 0011b - Bad_OpCode 0100b - I_Stack_Underflow 0101b - I_Stack_Overflow 0110b - D_Stack_Underflow 0111b - D_Stack_Overflow 1000b - Non-DMem_Sel 1001b - D_Parity_Error

Type	Bit No.	Bit Function	Bit Description
	23:20	Reserved	Reserved
	31:24	Reserved except for the following	00h - No Error 0Dh - MC_IMC_FORCE_SR_S3_TIMEOUT 0Eh - MC_CPD_UNCPD_ST_TIMEOUT 0Fh - MC_PKGS_SAFE_WP_TIMEOUT 43h - MC_PECI_MAILBOX_QUIESCE_TIMEOUT 5Ch - MC_MORE_THAN_ONE_LT_AGENT 60h - MC_INVALID_PKGS_REQ_PCH 61h - MC_INVALID_PKGS_REQ_QPI 62h - MC_INVALID_PKGS_RES_QPI 63h - MC_INVALID_PKGC_RES_PCH 64h - MC_INVALID_PKG_STATE_CONFIG 70h - MC_WATCHDG_TIMEOUT_PKGC_SECONDARY 71h - MC_WATCHDG_TIMEOUT_PKGC_MAIN 72h - MC_WATCHDG_TIMEOUT_PKGS_MAIN 7ah - MC_HA_FAILSTS_CHANGE_DETECTED 81h - MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	56:32	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.4.2 Intel QPI Machine Check Errors

Table 16-14. Intel QPI MC Error Codes for IA32_MC6_STATUS and IA32_MC7_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Bus error format: 1PPTRRRRIILL
Model specific errors			
	56:16	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.4.3 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC8_STATUS-IA32_MC11_STATUS. The supported error codes are follows the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture,"). MSR_ERROR_CONTROL.[bit 1] can enable additional informa-

tion logging of the IMC. The additional error information logged by the IMC is stored in IA32_MCi_STATUS and IA32_MCi_MISC; (i = 8, 11).

Table 16-15. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 8, 11)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Bus error format: 1PPTRRRRIILL
Model specific errors	31:16	Reserved except for the following	001H - Address parity error 002H - HA Wrt buffer Data parity error 004H - HA Wrt byte enable parity error 008H - Corrected patrol scrub error 010H - Uncorrected patrol scrub error 020H - Corrected spare error 040H - Uncorrected spare error
Model specific errors	36:32	Other info	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first device error when corrected error is detected during normal read.
	37	Reserved	Reserved
	56:38		See Chapter 15, "Machine-Check Architecture,"
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

Table 16-16. Intel IMC MC Error Codes for IA32_MCi_MISC (i= 8, 11)

Type	Bit No.	Bit Function	Bit Description
MCA addr info[1]	8:0		See Chapter 15, "Machine-Check Architecture,"
Model specific errors	13:9		▪ When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second device error when corrected error is detected during normal read. ▪ Otherwise contain parity error if MCi_Status indicates HA_WB_Data or HA_W_BE parity error.
Model specific errors	29:14	ErrMask_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error bit mask.
Model specific errors	45:30	ErrMask_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error bit mask.
	50:46	FailRank_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error failing rank.
	55:51	FailRank_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error failing rank.
	58:56	Reserved	Reserved
	61:59	Reserved	Reserved
	62	Valid_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data from the first correctable error in a memory device.
	63	Valid_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data due to a second correctable error in a memory device. Use this information only after there is valid first error info indicated by bit 62.

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.5 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR E5 V2 AND INTEL® XEON® PROCESSOR E7 V2 FAMILIES, MACHINE ERROR CODES FOR MACHINE CHECK

The Intel® Xeon® processor E5 v2 family and the Intel® Xeon® processor E7 v2 family are based on the Ivy Bridge-EP microarchitecture and can be identified with CPUID DisplayFamily_DisplaySignature 06_3EH. Incremental error codes for internal machine check error from PCU controller is reported in the register bank IA32_MC4, Table lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS. Incremental MC error codes related to the Intel QPI links are reported in the register banks IA32_MC5. Information listed in Table 16-14 for QPI MC error code apply to IA32_MC5_STATUS. Incremental error codes for the memory controller unit is reported in the register banks IA32_MC9-IA32_MC16. Table 16-18 lists model-specific error codes apply to IA32_MCi_STATUS, i = 9-16.

16.5.1 Internal Machine Check Errors

Table 16-17. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	
Model specific errors	19:16	Reserved except for the following	0000b - No Error 0001b - Non_IMem_Sel 0010b - I_Parity_Error 0011b - Bad_OpCode 0100b - I_Stack_Underflow 0101b - I_Stack_Overflow 0110b - D_Stack_Underflow 0111b - D_Stack_Overflow 1000b - Non-DMem_Sel 1001b - D_Parity_Error
	23:20	Reserved	Reserved
	31:24	Reserved except for the following	00h - No Error 0Dh - MC_IMC_FORCE_SR_S3_TIMEOUT 0Eh - MC_CPD_UNCPD_ST_TIMEOUT 0Fh - MC_PKGS_SAFE_WP_TIMEOUT 43h - MC_PECI_MAILBOX_QUIESCE_TIMEOUT 44h - MC_CRITICAL_VR_FAILED 45h - MC_ICC_MAX-NOTSUPPORTED 5Ch - MC_MORE_THAN_ONE_LT_AGENT 60h - MC_INVALID_PKGS_REQ_PCH 61h - MC_INVALID_PKGS_REQ_QPI 62h - MC_INVALID_PKGS_RES_QPI 63h - MC_INVALID_PKGC_RES_PCH 64h - MC_INVALID_PKG_STATE_CONFIG 70h - MC_WATCHDG_TIMEOUT_PKGC_SECONDARY 71h - MC_WATCHDG_TIMEOUT_PKGC_MAIN 72h - MC_WATCHDG_TIMEOUT_PKGS_MAIN

Type	Bit No.	Bit Function	Bit Description
			7Ah - MC_HA_FAILSTS_CHANGE_DETECTED 7Bh - MC_PCIE_R2PCIE-RW_BLOCK_ACK_TIMEOUT 81h - MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	56:32	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.5.2 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC9_STATUS-IA32_MC16_STATUS. The supported error codes are follows the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture").

MSR_ERROR_CONTROL.[bit 1] can enable additional information logging of the IMC. The additional error information logged by the IMC is stored in IA32_MCi_STATUS and IA32_MCi_MISC; (i = 9-16).

IA32_MCi_STATUS (i=9-12) log errors from the first memory controller. The second memory controller logs into IA32_MCi_STATUS (i=13-16).

Table 16-18. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 9-16)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Memory Controller error format: 000F 0000 1MMM CCCC
Model specific errors	31:16	Reserved except for the following	001H - Address parity error 002H - HA Wrt buffer Data parity error 004H - HA Wrt byte enable parity error 008H - Corrected patrol scrub error
			010H - Uncorrected patrol scrub error 020H - Corrected spare error 040H - Uncorrected spare error 080H - Corrected memory read error. (Only applicable with iMC's "Additional Error logging" Mode-1 enabled.) 100H - iMC, WDB, parity errors
	36:32	Other info	When MSR_ERROR_CONTROL.[1] is set, logs an encoded value from the first error device.
	37	Reserved	Reserved
	56:38		See Chapter 15, "Machine-Check Architecture,"
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

Table 16-19. Intel IMC MC Error Codes for IA32_MCi_MISC (i= 9-16)

Type	Bit No.	Bit Function	Bit Description
MCA addr info[1]	8:0		See Chapter 15, "Machine-Check Architecture,"
Model specific errors	13:9		If the error logged is MCWrDataPar error or MCWrBEPar error, this field is the WDB ID that has the parity error. OR if the second error logged is a correctable read error, MC logs the second error device in this field.
Model specific errors	29:14	ErrMask_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error bit mask.
Model specific errors	45:30	ErrMask_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error bit mask.
	50:46	FailRank_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error failing rank.
	55:51	FailRank_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error failing rank.
	61:56		Reserved
	62	Valid_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data from a correctable error from memory read associated with first error device.
	63	Valid_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data due to a second correctable error in a memory device. Use this information only after there is valid first error info indicated by bit 62.

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.5.3 Home Agent Machine Check Errors

Memory errors from the first memory controller may be logged in the IA32_MC7_{STATUS,ADDR,MISC} registers, while the second memory controller logs to IA32_MC8_{STATUS,ADDR,MISC}.

16.6 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR E5 V3 FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

The Intel® Xeon® processor E5 v3 family is based on the Haswell-E microarchitecture and can be identified with CPUID DisplayFamily_DisplaySignature 06_3FH. Incremental error codes for internal machine check error from PCU controller is reported in the register bank IA32_MC4, Table 16-20 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS. Incremental MC error codes related to the Intel QPI links are reported in the register banks IA32_MC5, IA32_MC20, and IA32_MC21. Information listed in Table 16-21 for QPI MC error codes. Incremental error codes for the memory controller unit is reported in the register banks IA32_MC9-IA32_MC16. Table 16-22 lists model-specific error codes apply to IA32_MCi_STATUS, i = 9-16.

16.6.1 Internal Machine Check Errors

Table 16-20. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	
MCACOD[2]	15:0	Internal Errors	0402h - PCU internal Errors 0403h - PCU internal Errors 0406h - Intel TXT Errors 0407h - Other UBOX internal Errors. On an IERR caused by a core 3-strike the IA32_MC3_STATUS (MLC) is copied to the IA32_MC4_STATUS (After a 3-strike, the core MCA banks will be unavailable).
Model specific errors	19:16	Reserved except for the following	0000b - No Error 00xxb - PCU internal error
	23:20	Reserved	Reserved
	31:24	Reserved except for the following	00h - No Error 09h - MC_MESSAGE_CHANNEL_TIMEOUT 13h - MC_DMI_TRAINING_TIMEOUT 15h - MC_DMI_CPU_RESET_ACK_TIMEOUT 1Eh - MC_VR_ICC_MAX_LT_FUSED_ICC_MAX 25h - MC_SVID_COMMAND_TIMEOUT 29h - MC_VR_VOUT_MAC_LT_FUSED_SVID 2Bh - MC_PKGC_WATCHDOG_HANG_CBZ_DOWN 2Ch - MC_PKGC_WATCHDOG_HANG_CBZ_UP 44h - MC_CRITICAL_VR_FAILED 46h - MC_VID_RAMP_DOWN_FAILED 49h - MC_SVID_WRITE_REG_VOUT_MAX_FAILED 4Bh - MC_BOOT_VID_TIMEOUT. Timeout setting boot VID for DRAM 0. 4Fh - MC_SVID_COMMAND_ERROR. 52h - MC_FIVR_CATAS_OVERVOL_FAULT. 53h - MC_FIVR_CATAS_OVERCUR_FAULT. 57h - MC_SVID_PKGC_REQUEST_FAILED 58h - MC_SVID_IMON_REQUEST_FAILED 59h - MC_SVID_ALERT_REQUEST_FAILED 62h - MC_INVALID_PKGS_RSP_QPI 64h - MC_INVALID_PKG_STATE_CONFIG 67h - MC_HA_IMC_RW_BLOCK_ACK_TIMEOUT 6Ah - MC_MSGCH_PMREQ_CMP_TIMEOUT 72h - MC_WATCHDG_TIMEOUT_PKGS_MASTER 81h - MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	56:32	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, “Machine-Check Architecture,” for more information.

2. The internal error codes may be model-specific.

16.6.2 Intel QPI Machine Check Errors

MC error codes associated with the Intel QPI agents are reported in the MSRs IA32_MC5_STATUS, IA32_MC20_STATUS, and IA32_MC21_STATUS. The supported error codes follow the architectural MCACOD definition type 1PPTRRRRIILL (see Chapter 15, "Machine-Check Architecture,").

Table 16-21 lists model-specific fields to interpret error codes applicable to IA32_MC5_STATUS, IA32_MC20_STATUS, and IA32_MC21_STATUS.

Table 16-21. Intel QPI MC Error Codes for IA32_MCi_STATUS (i = 5, 20, 21)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Bus error format: 1PPTRRRRIILL
Model specific errors	31:16	MSCOD	02h - Intel QPI physical layer detected drift buffer alarm. 03h - Intel QPI physical layer detected latency buffer rollover. 10h - Intel QPI link layer detected control error from R3QPI. 11h - Rx entered LLR abort state on CRC error. 12h - Unsupported or undefined packet. 13h - Intel QPI link layer control error. 15h - RBT used un-initialized value. 20h - Intel QPI physical layer detected a QPI in-band reset but aborted initialization 21h - Link failover data self-healing 22h - Phy detected in-band reset (no width change). 23h - Link failover clock failover 30h -Rx detected CRC error - successful LLR after Phy re-init. 31h -Rx detected CRC error - successful LLR without Phy re-init. All other values are reserved.
	37:32	Reserved	Reserved
	52:38	Corrected Error Cnt	
	56:53	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.6.3 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC9_STATUS-IA32_MC16_STATUS. The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture,").

MSR_ERROR_CONTROL.[bit 1] can enable additional information logging of the IMC. The additional error information logged by the IMC is stored in IA32_MCi_STATUS and IA32_MCi_MISC; (i = 9-16).

IA32_MCi_STATUS (i=9-12) log errors from the first memory controller. The second memory controller logs into IA32_MCi_STATUS (i=13-16).

Table 16-22. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 9-16)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model specific errors	31:16	Reserved except for the following	0001H - DDR3 address parity error 0002H - Uncorrected HA write data error
			0004H - Uncorrected HA data byte enable error 0008H - Corrected patrol scrub error 0010H - Uncorrected patrol scrub error 0020H - Corrected spare error 0040H - Uncorrected spare error 0080H - Corrected memory read error. (Only applicable with iMC's "Additional Error logging" Mode-1 enabled.) 0100H - iMC, write data buffer parity errors 0200H - DDR4 command address parity error
	36:32	Other info	When MSR_ERROR_CONTROL.[1] is set, logs an encoded value from the first error device.
	37	Reserved	Reserved
	56:38		See Chapter 15, "Machine-Check Architecture,"
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

Table 16-23. Intel IMC MC Error Codes for IA32_MCi_MISC (i= 9-16)

Type	Bit No.	Bit Function	Bit Description
MCA addr info[1]	8:0		See Chapter 15, "Machine-Check Architecture,"
Model specific errors	13:9		If the error logged is MCWrDataPar error or MCWrBEPar error, this field is the WDB ID that has the parity error. OR if the second error logged is a correctable read error, MC logs the second error device in this field.
Model specific errors	29:14	ErrMask_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error bit mask.
Model specific errors	45:30	ErrMask_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error bit mask.
	50:46	FailRank_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log first-device error failing rank.
	55:51	FailRank_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, allows the iMC to log second-device error failing rank.
	61:56		Reserved
	62	Valid_1stErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data from a correctable error from memory read associated with first error device.
	63	Valid_2ndErrDev	When MSR_ERROR_CONTROL.[1] is set, indicates the iMC has logged valid data due to a second correctable error in a memory device. Use this information only after there is valid first error info indicated by bit 62.

NOTES:
1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.6.4 Home Agent Machine Check Errors

Memory errors from the first memory controller may be logged in the IA32_MC7_{STATUS,ADDR,MISC} registers, while the second memory controller logs to IA32_MC8_{STATUS,ADDR,MISC}.

16.7 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR D FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

The Intel® Xeon® processor D family is based on the Broadwell microarchitecture and can be identified with CPUID DisplayFamily_DisplaySignature 06_56H. Incremental error codes for internal machine check error from PCU controller is reported in the register bank IA32_MC4, Table 16-24 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS. Incremental error codes for the memory controller unit is reported in the register banks IA32_MC9-IA32_MC10. Table 16-18 lists model-specific error codes apply to IA32_MCi_STATUS, i = 9-10.

16.7.1 Internal Machine Check Errors

Table 16-24. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	
MCACOD[2]	15:0	internal Errors	0402h - PCU internal Errors 0403h - internal Errors 0406h - Intel TXT Errors 0407h - Other UBOX internal Errors. On an IERR caused by a core 3-strike the IA32_MC3_STATUS (MLC) is copied to the IA32_MC4_STATUS (After a 3-strike, the core MCA banks will be unavailable).
Model specific errors	19:16	Reserved except for the following	0000b - No Error 00x1b - PCU internal error 001xb - PCU internal error

Type	Bit No.	Bit Function	Bit Description
	23:20	Reserved except for the following	x1xxb - UBOX error
	31:24	Reserved except for the following	00h - No Error 09h - MC_MESSAGE_CHANNEL_TIMEOUT 13h - MC_DMI_TRAINING_TIMEOUT 15h - MC_DMI_CPU_RESET_ACK_TIMEOUT 1Eh - MC_VR_ICC_MAX_LT_FUSED_ICC_MAX 25h - MC_SVID_COMMAND_TIMEOUT 26h - MCA_PKGC_DIRECT_WAKE_RING_TIMEOUT 29h - MC_VR_VOUT_MAC_LT_FUSED_SVID 2Bh - MC_PKGC_WATCHDOG_HANG_CBZ_DOWN 2Ch - MC_PKGC_WATCHDOG_HANG_CBZ_UP 44h - MC_CRITICAL_VR_FAILED 46h - MC_VID_RAMP_DOWN_FAILED 49h - MC_SVID_WRITE_REG_VOUT_MAX_FAILED 4Bh - MC_PP1_BOOT_VID_TIMEOUT. Timeout setting boot VID for DRAM 0. 4Fh - MC_SVID_COMMAND_ERROR. 52h - MC_FIVR_CATAS_OVERVOL_FAULT. 53h - MC_FIVR_CATAS_OVERCUR_FAULT. 57h - MC_SVID_PKGC_REQUEST_FAILED 58h - MC_SVID_IMON_REQUEST_FAILED 59h - MC_SVID_ALERT_REQUEST_FAILED 62h - MC_INVALID_PKGS_RSP_QPI 64h - MC_INVALID_PKG_STATE_CONFIG 67h - MC_HA_IMC_RW_BLOCK_ACK_TIMEOUT 6Ah - MC_MSGCH_PMREQ_CMP_TIMEOUT 72h - MC_WATCHDG_TIMEOUT_PKGS_MASTER 81h - MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	56:32	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.
2. The internal error codes may be model-specific.

16.7.2 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC9_STATUS-IA32_MC10_STATUS. The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture,").

MSR_ERROR_CONTROL.[bit 1] can enable additional information logging of the IMC. The additional error information logged by the IMC is stored in IA32_MCi_STATUS and IA32_MCi_MISC; (i = 9-10).

Table 16-25. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 9-10)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model specific errors	31:16	Reserved except for the following	0001H - DDR3 address parity error 0002H - Uncorrected HA write data error 0004H - Uncorrected HA data byte enable error 0008H - Corrected patrol scrub error 0010H - Uncorrected patrol scrub error 0100H - iMC, write data buffer parity errors 0200H - DDR4 command address parity error
	36:32	Other info	Reserved
	37	Reserved	Reserved
	56:38		See Chapter 15, "Machine-Check Architecture,"
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.8 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR E5 V4 FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

The Intel® Xeon® processor E5 v4 family is based on the Broadwell microarchitecture and can be identified with CPUID DisplayFamily_DisplaySignature 06_4FH. Incremental error codes for internal machine check error from PCU controller is reported in the register bank IA32_MC4, Table 16-20 in Section 16.6.1lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS.

Incremental MC error codes related to the Intel QPI links are reported in the register banks IA32_MC5, IA32_MC20, and IA32_MC21. Information listed in Table 16-21 of Section 16.6.1 covers QPI MC error codes.

16.8.1 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC9_STATUS-IA32_MC16_STATUS. The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture").

Table 16-26 lists model-specific error codes apply to IA32_MCi_STATUS, i = 9-16.

IA32_MCi_STATUS (i=9-12) log errors from the first memory controller. The second memory controller logs into IA32_MCi_STATUS (i=13-16).

Table 16-26. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 9-16)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model specific errors	31:16	Reserved except for the following	0001H - DDR3 address parity error 0002H - Uncorrected HA write data error 0004H - Uncorrected HA data byte enable error 0008H - Corrected patrol scrub error 0010H - Uncorrected patrol scrub error 0020H - Corrected spare error 0040H - Uncorrected spare error 0100H - iMC, write data buffer parity errors 0200H - DDR4 command address parity error
	36:32	Other info	Reserved
	37	Reserved	Reserved
	56:38		See Chapter 15, "Machine-Check Architecture,"
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.8.2 Home Agent Machine Check Errors

MC error codes associated with mirrored memory corrections are reported in the MSRs IA32_MC7_MISC and IA32_MC8_MISC. Table 16-27 lists model-specific error codes apply to IA32_MCi_MISC, i = 7, 8.

Memory errors from the first memory controller may be logged in the IA32_MC7_{STATUS,ADDR,MISC} registers, while the second memory controller logs to IA32_MC8_{STATUS,ADDR,MISC}.

Table 16-27. Intel HA MC Error Codes for IA32_MCi_MISC (i= 7, 8)

Bit No.	Bit Function	Bit Description
5:0	LSB	See Figure 15-8.
8:6	Address Mode	See Table 15-3.
40:9	Reserved	Reserved
41	Failover	Error occurred at a pair of mirrored memory channels. Error was corrected by mirroring with channel failover.
42	Mirrorcorr	Error was corrected by mirroring and primary channel scrubbed successfully.
63:43	Reserved	Reserved

16.9 INCREMENTAL DECODING INFORMATION: INTEL® XEON® PROCESSOR SCALABLE FAMILY, MACHINE ERROR CODES FOR MACHINE CHECK

In the Intel® Xeon® Processor Scalable Family with CPUID DisplayFamily_DisplaySignature 06_55H, incremental error codes for internal machine check errors from the PCU controller are reported in the register bank IA32_MC4. Table 16-28 in Section 16.9.1 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS.

16.9.1 Internal Machine Check Errors

Table 16-28. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	
MCACOD[2]	15:0	Internal Errors	0402h - PCU internal Errors 0403h - PCU internal Errors 0406h - Intel TXT Errors 0407h - Other UBOX internal Errors. On an IERR caused by a core 3-strike the IA32_MC3_STATUS (MLC) is copied to the IA32_MC4_STATUS (After a 3-strike, the core MCA banks will be unavailable).
Model specific errors	19:16	Reserved except for the following	0000b - No Error 00xxb - PCU internal error
	23:20	Reserved	Reserved
	31:24	Reserved except for the following	00h - No Error 0Dh - MCA_DMI_TRAINING_TIMEOUT 0Fh - MCA_DMI_CPU_RESET_ACK_TIMEOUT 10h - MCA_MORE_THAN_ONE_LT_AGENT 1Eh - MCA_BIOS_RST_CPL_INVALID_SEQ 1Fh - MCA_BIOS_INVALID_PKG_STATE_CONFIG 25h - MCA_MESSAGE_CHANNEL_TIMEOUT 27h - MCA_MSGCH_PMREQ_CMP_TIMEOUT 30h - MCA_PKGC_DIRECT_WAKE_RING_TIMEOUT 31h - MCA_PKGC_INVALID_RSP_PCH 33h - MCA_PKGC_WATCHDOG_HANG_CBZ_DOWN 34h - MCA_PKGC_WATCHDOG_HANG_CBZ_UP 38h - MCA_PKGC_WATCHDOG_HANG_C3_UP_SF 40h - MCA_SVID_VCCIN_VR_ICC_MAX_FAILURE 41h - MCA_SVID_COMMAND_TIMEOUT 42h - MCA_SVID_VCCIN_VR_VOUT_MAX_FAILURE 43h - MCA_SVID_CPU_VR_CAPABILITY_ERROR 44h - MCA_SVID_CRITICAL_VR_FAILED 45h - MCA_SVID_SA_ITD_ERROR 46h - MCA_SVID_READ_REG_FAILED 47h - MCA_SVID_WRITE_REG_FAILED 48h - MCA_SVID_PKGC_INIT_FAILED

Type	Bit No.	Bit Function	Bit Description
			49h - MCA_SVID_PKGC_CONFIG_FAILED 4Ah - MCA_SVID_PKGC_REQUEST_FAILED 4Bh - MCA_SVID_IMON_REQUEST_FAILED 4Ch - MCA_SVID_ALERT_REQUEST_FAILED 4Dh - MCA_SVID_MCP_VP_ABSENT_OR_RAMP_ERROR 4Eh - MCA_SVID_UNEXPECTED_MCP_VP_DETECTED 51h - MCA_FIVR_CATAS_OVERVOL_FAULT 52h - MCA_FIVR_CATAS_OVERCUR_FAULT 58h - MCA_WATCHDG_TIMEOUT_PKGC_SECONDARY 59h - MCA_WATCHDG_TIMEOUT_PKGC_MAIN 5Ah - MCA_WATCHDG_TIMEOUT_PKGS_MAIN 61h - MCA_PKGS_CPD_UNPCD_TIMEOUT 63h - MCA_PKGS_INVALID_REQ_PCH 64h - MCA_PKGS_INVALID_REQ_INTERNAL 65h - MCA_PKGS_INVALID_RSP_INTERNAL 6Bh - MCA_PKGS_SMBUS_VPP_PAUSE_TIMEOUT 81h - MC_RECOVERABLE_DIE_THERMAL_TOO_HOT
	52:32	Reserved	Reserved
	54:53	CORR_ERR_STATUS	Reserved
	56:55	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.
2. The internal error codes may be model-specific.

16.9.2 Interconnect Machine Check Errors

MC error codes associated with the link interconnect agents are reported in the MSRs IA32_MC5_STATUS, IA32_MC12_STATUS, IA32_MC19_STATUS. The supported error codes follow the architectural MCACOD definition type 1PPTRRRRIILL (see Chapter 15, "Machine-Check Architecture").

Table 16-29 lists model-specific fields to interpret error codes applicable to IA32_MCi_STATUS, i= 5, 12, 19.

Table 16-29. Interconnect MC Error Codes for IA32_MCi_STATUS, i = 5, 12, 19

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Bus error format: 1PPTRRRRIILL The two supported compound error codes: - 0x0C0F - Unsupported/Undefined Packet - 0x0E0F - For all other corrected and uncorrected errors

Type	Bit No.	Bit Function	Bit Description
Model specific errors	21:16	MSCOD	The encoding of Uncorrectable (UC) errors are: 00h - UC Phy Initialization Failure. 01h - UC Phy detected drift buffer alarm. 02h - UC Phy detected latency buffer rollover. 10h - UC link layer Rx detected CRC error: unsuccessful LLR entered abort state 11h - UC LL Rx unsupported or undefined packet. 12h - UC LL or Phy control error. 13h - UC LL Rx parameter exchange exception. 1fh - UC LL detected control error from the link-mesh interface The encoding of correctable (COR) errors are: 20h - COR Phy initialization abort 21h - COR Phy reset 22h - COR Phy lane failure, recovery in x8 width. 23h - COR Phy L0c error corrected without Phy reset 24h - COR Phy L0c error triggering Phy reset 25h - COR Phy L0p exit error corrected with Phy reset 30h - COR LL Rx detected CRC error - successful LLR without Phy re-init. 31h - COR LL Rx detected CRC error - successful LLR with Phy re-init. All other values are reserved.
	31:22	MSCOD_SPARE	The definition below applies to MSCOD 12h (UC LL or Phy Control Errors) [Bit 22] : Phy Control Error [Bit 23] : Unexpected Retry.Ack flit [Bit 24] : Unexpected Retry.Req flit [Bit 25] : RF parity error [Bit 26] : Routeback Table error [Bit 27] : unexpected Tx Protocol flit (EOP, Header or Data) [Bit 28] : Rx Header-or-Credit BGF credit overflow/underflow [Bit 29] : Link Layer Reset still in progress when Phy enters L0 (Phy training should not be enabled until after LL reset is complete as indicated by KTILCL.LinkLayerReset going back to 0). [Bit 30] : Link Layer reset initiated while protocol traffic not idle [Bit 31] : Link Layer Tx Parity Error
	37:32	Reserved	Reserved
	52:38	Corrected Error Cnt	
	56:53	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.9.3 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC13_STATUS-IA32_MC18_STATUS. The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture").

IA32_MCi_STATUS (i=13,14,17) log errors from the first memory controller. The second memory controller logs into IA32_MCi_STATUS (i=15,16,18).

Table 16-30. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 13-18)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model specific errors	31:16	Reserved except for the following	0001H - Address parity error 0002H - HA write data parity error 0004H - HA write byte enable parity error 0008H - Corrected patrol scrub error 0010H - Uncorrected patrol scrub error 0020H - Corrected spare error 0040H - Uncorrected spare error 0080H - Any HA read error 0100H - WDB read parity error 0200H - DDR4 command address parity error 0400H - Uncorrected address parity error 0800H - Unrecognized request type 0801H - Read response to an invalid scoreboard entry 0802H - Unexpected read response 0803H - DDR4 completion to an invalid scoreboard entry 0804H - Completion to an invalid scoreboard entry 0805H - Completion FIFO overflow 0806H - Correctable parity error 0807H - Uncorrectable error 0808H - Interrupt received while outstanding interrupt was not ACKed 0809H - ERID FIFO overflow 080aH - Error on Write credits 080bH - Error on Read credits 080cH - Scheduler error 080dH - Error event
	36:32	Other info	MC logs the first error device. This is an encoded 5-bit value of the device.
	37	Reserved	Reserved
	56:38		See Chapter 15, "Machine-Check Architecture,"
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.9.4 M2M Machine Check Errors

MC error codes associated with M2M are reported in the MSRs IA32_MC7_STATUS, IA32_MC8_STATUS. The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture,").

Table 16-31. M2M MC Error Codes for IA32_MCi_STATUS (i= 7-8)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Compound error format: 0000 0000 1MMM CCCC
Model specific errors	16	MscodDataRdErr	Logged an MC read data error
	17	Reserved	Reserved
	18	MscodPtlWrErr	Logged an MC partial write data error
	19	MscodFullWrErr	Logged a full write data error
	20	MscodBgfErr	Logged an M2M clock-domain-crossing buffer (BGF) error
	21	MscodTimeOut	Logged an M2M time out
	22	MscodParErr	Logged an M2M tracker parity error
	23	MscodBucket1Err	Logged a fatal Bucket1 error
	31:24	Reserved	Reserved
	36:32	Other info	MC logs the first error device. This is an encoded 5-bit value of the device.
	37	Reserved	Reserved
	56:38		See Chapter 15, "Machine-Check Architecture,"
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.9.5 Home Agent Machine Check Errors

MC error codes associated with mirrored memory corrections are reported in the MSRs IA32_MC7_MISC and IA32_MC8_MISC. Table 16-32 lists model-specific error codes apply to IA32_MCi_MISC, i = 7, 8.

Memory errors from the first memory controller may be logged in the IA32_MC7_{STATUS,ADDR,MISC} registers, while the second memory controller logs to IA32_MC8_{STATUS,ADDR,MISC}.

Table 16-32. Intel HA MC Error Codes for IA32_MCi_MISC (i= 7, 8)

Bit No.	Bit Function	Bit Description
5:0	LSB	See Figure 15-8.
8:6	Address Mode	See Table 15-3.
40:9	Reserved	Reserved
61:41	Reserved	Reserved
62	Mirrorcorr	Error was corrected by mirroring and primary channel scrubbed successfully.
63	Failover	Error occurred at a pair of mirrored memory channels. Error was corrected by mirroring with channel failover.

16.10 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY WITH CPUID DISPLAYFAMILY_DISPLAYMODEL SIGNATURE 06_5FH, MACHINE ERROR CODES FOR MACHINE CHECK

In Intel Atom® processors based on Goldmont Microarchitecture with CPUID DisplayFamily_DisplaySignature 06_5FH (Denverton), incremental error codes for the memory controller unit are reported in the register banks IA32_MC6 and IA32_MC7. Table 16-33 in Section 16.10.1 lists model-specific fields to interpret error codes applicable to IA32_MCi_STATUS, i = 6, 7.

16.10.1 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC6_STATUS and IA32_MC7_STATUS. The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture").

Table 16-33. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 6, 7)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	
Model specific errors	31:16	Reserved except for the following	01h - Cmd/Addr parity 02h - Corrected Demand/Patrol Scrub Error 04h - Uncorrected patrol scrub error 08h - Uncorrected demand read error 10h - WDB read ECC
	36:32	Other info	
	37	Reserved	
	56:38		See Chapter 15, "Machine-Check Architecture".
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.11 INCREMENTAL DECODING INFORMATION: 3RD GENERATION INTEL® XEON® PROCESSOR SCALABLE FAMILY WITH CPUID DISPLAYFAMILY_DISPLAYMODEL SIGNATURES 06_6AH AND 06_6CH, MACHINE ERROR CODES FOR MACHINE CHECK

In the 3rd generation Intel® Xeon® Processor Scalable Family with CPUID DisplayFamily_DisplaySignature of 06_6AH and 06_6CH, incremental error codes for internal machine check errors from the PCU controller are reported in the register bank IA32_MC4. Table 16-34 in Section 16.11.1 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS.

16.11.1 Internal Machine Check Errors

Table 16-34. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
Machine Check Error Codes[1]	15:0	MCCOD	
MCCOD	15:0	Internal Errors	The value of this field will be 0402h for the PCU and 0406h for internal firmware errors. This applies for any logged error.
Model specific errors	19:16	Reserved except for the following	Model specific error code bits 19:16. This logs the type of HW UC (PCU/VCU) error that has occurred. There are 7 errors defined. 01h - Instruction address out of valid space. 02h - Double bit RAM error on Instruction Fetch. 03h - Invalid OpCode seen. 04h - Stack Underflow. 05h - Stack Overflow. 06h - Data address out of valid space. 07h - Double bit RAM error on Data Fetch.
	23:20	Reserved except for the following	Model specific error code bits 23:20. This logs the type of HW FSM error that has occurred. There are 3 errors defined. 04h - Clock/power IP response timeout. 05h - SMBus controller raised SMI. 09h - PM controller received invalid transaction.
	31:24	Reserved except for the following	0Dh - MCA_LLC_BIST_ACTIVE_TIMEOUT 0Eh - MCA_DMI_TRAINING_TIMEOUT 0Fh - MCA_DMI_STRAP_SET_ARRIVAL_TIMEOUT 10h - MCA_DMI_CPU_RESET_ACK_TIMEOUT 11h - MCA_MORE_THAN_ONE_LT_AGENT 14h - MCA_INCOMPATIBLE_PCH_TYPE 1Eh - MCA_BIOS_RST_CPL_INVALID_SEQ 1Fh - MCA_BIOS_INVALID_PKG_STATE_CONFIG 2Dh - MCA_PCU_PMAX_CALIB_ERROR 2Eh - MCA_TSC100_SYNC_TIMEOUT 3Ah - MCA_GPSB_TIMEOUT 3Bh - MCA_PMSB_TIMEOUT 3Eh - MCA_IOSFSB_PMREQ_CMP_TIMEOUT 40h - MCA_SVID_VCCIN_VR_ICC_MAX_FAILURE 42h - MCA_SVID_VCCIN_VR_VOUT_FAILURE 43h - MCA_SVID_CPU_VR_CAPABILITY_ERROR 44h - MCA_SVID_CRITICAL_VR_FAILED 45h - MCA_SVID_SA_ITD_ERROR 46h - MCA_SVID_READ_REG_FAILED 47h - MCA_SVID_WRITE_REG_FAILED 4Ah - MCA_SVID_PKGC_REQUEST_FAILED

Type	Bit No.	Bit Function	Bit Description
			4Bh - MCA_SVID_IMON_REQUEST_FAILED 4Ch - MCA_SVID_ALERT_REQUEST_FAILED 4Dh - MCA_SVID_MCP_VR_RAMP_ERROR 56h - MCA_FIVR_PD_HARDERR 58h - MCA_WATCHDOG_TIMEOUT_PKGC_SECONDARY 59h - MCA_WATCHDOG_TIMEOUT_PKGC_MAIN 5Ah - MCA_WATCHDOG_TIMEOUT_PKGS_MAIN 5Bh - MCA_WATCHDOG_TIMEOUT_MSG_CH_FSM 5Ch - MCA_WATCHDOG_TIMEOUT_BULK_CR_FSM 5Dh - MCA_WATCHDOG_TIMEOUT_IOSFSB_FSM 60h - MCA_PKGS_SAFE_WP_TIMEOUT 61h - MCA_PKGS_CPD_UNCPD_TIMEOUT 62h - MCA_PKGS_INVALID_REQ_PCH 63h - MCA_PKGS_INVALID_REQ_INTERNAL 64h - MCA_PKGS_INVALID_RSP_INTERNAL 65h-7Ah - MCA_PKGS_RESET_PREP_TIMEOUT 7Bh - MCA_PKGS_SMBUS_VPP_PAUSE_TIMEOUT 7Ch - MCA_PKGS_SMBUS_MCP_PAUSE_TIMEOUT 7Dh - MCA_PKGS_SMBUS_SPD_PAUSE_TIMEOUT 80h - MCA_PKGC_DISP_BUSY_TIMEOUT 81h - MCA_PKGC_INVALID_RSP_PCH 83h - MCA_PKGC_WATCHDOG_HANG_CBZ_DOWN 84h - MCA_PKGC_WATCHDOG_HANG_CBZ_UP 87h - MCA_PKGC_WATCHDOG_HANG_C2_BLKMASTER 88h - MCA_PKGC_WATCHDOG_HANG_C2_PSLIMIT 89h - MCA_PKGC_WATCHDOG_HANG_SETDISP 8Bh - MCA_PKGC_ALLOW_L1_ERROR 90h - MCA_RECOVERABLE_DIE_THERMAL_TOO_HOT A0h - MCA_ADR_SIGNAL_TIMEOUT A1h - MCA_BCLK_FREQ_OC_ABOVE_THRESHOLD B0h - MCA_DISPATCHER_RUN_BUSY_TIMEOUT
	37:32	ENH_MCA_AVAIL0	Available when Enhanced MCA is in use.
	52:38	CORR_ERR_COUNT	Correctable error count.
	54:53	CORRERRORSTATUSIND	These bits are used to indicate when the number of corrected errors has exceeded the safe threshold to the point where an uncorrected error has become more likely to happen. Table 3 shows the encoding of these bits.
	56:55	ENH_MCA_AVAIL1	Available when Enhanced MCA is in use.
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, “Machine-Check Architecture,” for more information.

16.11.2 Interconnect Machine Check Errors

MC error codes associated with the link interconnect agents are reported in the MSRs IA32_MC5_STATUS, IA32_MC7_STATUS, IA32_MC8_STATUS. The supported error codes follow the architectural MCACOD definition type 1PPTRRRRIILL (see Chapter 15, "Machine-Check Architecture").

NOTE

The interconnect machine check errors in this section apply only to the 3rd generation Intel Xeon Processor Scalable Family with a CPUID DisplayFamily_DisplaySignature of 06_6AH. These do not apply to the 3rd generation Intel Xeon Processor Scalable Family with a CPUID DisplayFamily_DisplaySignature of 06_6CH.

Table 16-35 lists model-specific fields to interpret error codes applicable to IA32_MCi_STATUS, i= 5, 7, 8.

Table 16-35. Interconnect MC Error Codes for IA32_MCi_STATUS, i = 5, 7, 8

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Bus error format: 1PPTRRRRIILL The two supported compound error codes: - 0x0C0F - Unsupported/Undefined Packet. - 0x0E0F - For all other corrected and uncorrected errors.
Model specific errors	21:16	MSCOD	The encoding of Uncorrectable (UC) errors are: 00h - Phy Initialization Failure (NumInit). 01h - Phy Detected Drift Buffer Alarm. 02h - Phy Detected Latency Buffer Rollover. 10h - LL Rx detected CRC error: unsuccessful LLR (entered Abort state). 11h - LL Rx Unsupported/Undefined packet. 12h - LL or Phy Control Error. 13h - LL Rx Parameter Exception. 1Fh - LL Detected Control Error. The encoding of correctable (COR) errors are: 20h - Phy Initialization Abort. 21h - Phy Inband Reset. 22h - Phy Lane failure, recovery in x8 width. 23h - Phy L0c error corrected without Phy reset. 24h - Phy L0c error triggering Phy reset. 25h - Phy L0p exit error corrected with reset. 30h - LL Rx detected CRC error: successful LLR without Phy Reinit. 31h - LL Rx detected CRC error: successful LLR with Phy Reinit. 32h - Tx received LLR. All other values are reserved.

Type	Bit No.	Bit Function	Bit Description
	31:22	MSCOD_SPARE	The definition below applies to MSCOD 12h (UC LL or Phy Control Errors). [Bit 22] : Phy Control Error. [Bit 23] : Unexpected Retry.Ack flit. [Bit 24] : Unexpected Retry.Req flit. [Bit 25] : RF parity error. [Bit 26] : Routeback Table error. [Bit 27] : Unexpected Tx Protocol flit (EOP, Header or Data). [Bit 28] : Rx Header-or-Credit BGF credit overflow/underflow. [Bit 29] : Link Layer Reset still in progress when Phy enters L0 (Phy training should not be enabled until after LL reset is complete as indicated by KTILCL.LinkLayerReset going back to 0). [Bit 30] : Link Layer reset initiated while protocol traffic not idle. [Bit 31] : Link Layer Tx Parity Error.
	37:32	OTHER_INFO	Other Info.
	56:38	Corrected Error Cnt	See Chapter 15, "Machine-Check Architecture".
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

16.11.3 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers for the 3rd generation Intel® Xeon® Processor Scalable Family based on Ice Lake microarchitecture are defined in Table 16-37.

The MSRs reporting MC error codes differ depending on the CPUID DisplayFamily_DisplaySignature of the processor. See Table 16-36 for details.

Table 16-36. MSRs Reporting MC Error Codes by CPUID DisplayFamily_DisplaySignature

Processor	CPUID DisplayFamily_DisplaySignature	MSRs Reporting MC Error Codes
3rd generation Intel® Xeon® Processor Scalable Family based on Ice Lake microarchitecture	06_6AH	IA32_MC13_STATUS - IA32_MC15_STATUS IA32_MC17_STATUS - IA32_MC19_STATUS IA32_MC21_STATUS - IA32_MC23_STATUS IA32_MC25_STATUS - IA32_MC27_STATUS
3rd generation Intel® Xeon® Processor Scalable Family based on Ice Lake microarchitecture	06_6CH	IA32_MC13_STATUS - IA32_MC15_STATUS IA32_MC17_STATUS - IA32_MC19_STATUS

The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture").

Table 16-37. Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 13-15, 17-19, 21-23, 25-27)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Memory Controller error format: 0000 0000 1MMM CCCC
Model specific errors	31:16	Reserved except for the following	0001H - Address parity error. 0002H - Data parity error. 0003H - Data ECC error. 0004H - Data byte enable parity error. 0007H - Transaction ID parity error. 0008H - Corrected patrol scrub error. 0010H - Uncorrected patrol scrub error. 0020H - Corrected spare error. 0040H - Uncorrected spare error. 0080H - Corrected read error. 00A0H - Uncorrected read error. 00C0H - Uncorrected metadata. 0100H - WDB read parity error. 0103H - RPA parity error. 0104H - RPA parity error. 0105H - WPA parity error. 0106H - DDR_T_DPPP data BE error. 0107H - DDR_T_DPPP data error. 0108H - DDR link failure. 0111H - PCLS CAM error. 0112H - PCLS data error. 0200H - DDR4 command / address parity error. 0220H - HBM command / address parity error. 0221H - HBM data parity error. 0400H - RPQ parity (primary) error. 0401H - RPQ parity (buddy) error. 0404H - WPQ parity (primary) error. 0405H - WPQ parity (buddy) error. 0408H - RPB parity (primary) error. 0409H - RPB parity (buddy) error. 0800H - DDR-T bad request. 0801H - DDR Data response to an invalid entry. 0802H - DDR data response to an entry not expecting data. 0803H - DDR4 completion to an invalid entry. 0804H - DDR-T completion to an invalid entry. 0805H - DDR data/completion FIFO overflow. 0806H - DDR-T ERID correctable parity error. 0807H - DDR-T ERID uncorrectable error. 0808H - DDR-T interrupt received while outstanding interrupt was not ACKed.

Type	Bit No.	Bit Function	Bit Description
			0809H - ERID FIFO overflow. 080AH - DDR-T error on FNV write credits. 080BH - DDR-T error on FNV read credits. 080CH -DDR-T scheduler error. 080DH - DDR-T FNV error event. 080EH - DDR-T FNV thermal event. 080FH - CMI packet while idle. 0810H - DDR_T_RPQ_REQ_PARITY_ERR. 0811H - DDR_T_WPQ_REQ_PARITY_ERR. 0812H - 2LM_NMFILLWR_CAM_ERR. 0813H - CMI_CREDIT_OVERSUB_ERR. 0814H - CMI_CREDIT_TOTAL_ERR. 0815H - CMI_CREDIT_RSVD_POOL_ERR. 0816H - DDR_T_RD_ERROR. 0817H - WDB_FIFO_ERR. 0818H - CMI_REQ_FIFO_OVERFLOW. 0819H - CMI_REQ_FIFO_UNDERFLOW. 081AH - CMI_RSP_FIFO_OVERFLOW. 081BH - CMI_RSP_FIFO_UNDERFLOW. 081CH - CMI_MISC_MC_CRDT_ERRORS. 081DH - CMI_MISC_MC_ARB_ERRORS. 081EH - DDR_T_WR_CMPL_FIFO_OVERFLOW. 081FH - DDR_T_WR_CMPL_FIFO_UNDERFLOW. 0820H - CMI_RD_CPL_FIFO_OVERFLOW. 0821H - CMI_RD_CPL_FIFO_UNDERFLOW. 0822H - TME_KEY_PAR_ERR. 0823H - TME_CMI_MISC_ERR. 0824H - TME_CMI_OVFL_ERR. 0825H - TME_CMI_UFL_ERR. 0826H - TME_TEM_SECURE_ERR. 0827H - TME_UFILL_PAR_ERR.
	37:32	Other info	Other Info.
	56:38		See Chapter 15, “Machine-Check Architecture”.
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, “Machine-Check Architecture,” for more information.

Additional information is reported in the MSRs IA32_MC13_MISC - IA32_MC15_MISC, IA32_MC17_MISC - IA32_MC19_MISC, IA32_MC21_MISC - IA32_MC23_MISC, and IA32_MC25_MISC - IA32_MC27_MISC. Table 16-38 lists the information reported in IA32_MCi_MISC, i = 13-15, 17-19, 21-23, and 25-27.

Table 16-38. Additional Information Reported in IA32_MCi_MISC (i= 13-15, 17-19, 21-23, 25-27)

Bit No.	Bit Function	Bit Description
5:0	LSB	See Figure 15-8.
8:6	Address Mode	See Table 15-3.
18:9	Column	Component of sub-DIMM address. Bits 18-17: Reserved Bit 16: Column 9 Bit 15: Column 8 Bit 14: Column 7 Bit 13: Column 6 Bit 12: Column 5 Bit 11: Column 4 Bit 10: Column 3 Bit 9: Reserved
39:19	Row	Component of sub-DIMM address.
45:40	Bank	Component of sub-DIMM address. Bit 45: Reserved Bit 44: Bank group 2 Bit 43: Bank address 1 Bit 42: Bank address 0 Bit 41: Bank group 1 Bit 40: Bank address 0
51:46	Failed Device	Failing device for correctable error (not valid for uncorrectable or transient errors).
55:52	SubRank	Logical Rank (3D Stacked SDRAM)
58:56	Rank	Rank
62:59	ECC Mode	0000b: SDDC memory mode 0001b: SDDC 0100b: ADDDC memory mode 0101b: ADDDC 1000b: Read from DDRT Other values: Reserved
63	Transient	0b: 1b: Error was transient

16.11.4 M2M Machine Check Errors

MC error codes associated with M2M for the 3rd generation Intel Xeon Processor Scalable Family with a CPUID DisplayFamily_DisplaySignature of 06_6AH are reported in the MSRs IA32_MC12_STATUS, IA32_MC16_STATUS, IA32_MC20_STATUS, and IA32_MC24_STATUS.

MC error codes associated with M2M for the 3rd generation Intel Xeon Processor Scalable Family with a CPUID DisplayFamily_DisplaySignature of 06_6CH are reported in the MSRs IA32_MC12_STATUS and IA32_MC16_STATUS.

The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture").

Table 16-39. M2M MC Error Codes for IA32_MCi_STATUS (i= 12, 16, 20, 24)

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0	MCACOD	Compound error format: 0000 0000 1MMM CCCC
Model specific errors	16	MscodDataRdErr	Logged an MC read data error.
	17	Reserved	Reserved
	18	MscodPtlWrErr	Logged an MC partial write data error.
	19	MscodFullWrErr	Logged a full write data error.
	20	MscodBgfErr	Logged an M2M clock-domain-crossing buffer (BGF) error.
	21	MscodTimeOut	Logged an M2M time out.
	22	MscodParErr	Logged an M2M tracker parity error.
	23	MscodBucket1Err	Logged a fatal Bucket1 error.
	25:24	MscodDDRType	Logged a DDR/DDR-T specific error.
	31:26	MscodMiscErrs	Logged a miscellaneous error.
	37:32	Other info	Other Info.
	56:38		See Chapter 15, "Machine-Check Architecture".
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

MC error codes associated with mirrored memory corrections are reported in the MSRs IA32_MC12_MISC, IA32_MC16_MISC, IA32_MC20_MISC, and IA32_MC24_MISC. The model-specific error codes listed in Table 16-32 also apply to IA32_MCi_MISC, i = 12, 16, 20, 24.

16.12 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY WITH CPUID DISPLAYFAMILY_DISPLAYMODEL SIGNATURE 06_86H, MACHINE ERROR CODES FOR MACHINE CHECK

In Intel Atom® processors based on Tremont microarchitecture with CPUID DisplayFamily_DisplaySignature 06_86H, incremental error codes for internal machine check errors from the PCU controller are reported in the register bank IA32_MC4. Table 16-34 in Section 16.11.1 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS.

16.12.1 Integrated Memory Controller Machine Check Errors

MC error codes associated with integrated memory controllers are reported in the MSRs IA32_MC13_STATUS - IA32_MC15_STATUS. The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture").

The IA32_MCi_STATUS MSR (where i = 13, 14, 15) contains information related to a machine check error if its VAL(valid) flag is set. Bit definitions are the same as those found in Table 16-37 "*Intel IMC MC Error Codes for IA32_MCi_STATUS (i= 13-15, 17-19, 21-23, 25-27)*".

The IA32_MCi_MISC MSR (where i = 13, 14, 15) contains information related memory corrections. Bit definitions are the same as those found in Table 16-38 "*Additional Information Reported in IA32_MCi_MISC (i= 13-15, 17-19, 21-23, 25-27)*".

16.12.2 M2M Machine Check Errors

MC error codes associated with M2M are reported in the IA32_MC12_STATUS MSR. The supported error codes follow the architectural MCACOD definition type 1MMMCCCC (see Chapter 15, "Machine-Check Architecture").

Bit definitions are the same as those found in Table 16-39 "*M2M MC Error Codes for IA32_MCi_STATUS (i= 12, 16, 20, 24)*".

16.13 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY 0FH MACHINE ERROR CODES FOR MACHINE CHECK

Table 16-40 provides information for interpreting additional family 0FH model-specific fields for external bus errors. These errors are reported in the IA32_MCi_STATUS MSRs. They are reported architecturally) as compound errors with a general form of *0000 1PPT RRRR IILL* in the MCA error code field. See Chapter 15 for information on the interpretation of compound error codes.

Table 16-40. Incremental Decoding Information: Processor Family 0FH Machine Error Codes For Machine Check

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0		
Model-specific error codes	16	FSB address parity	Address parity error detected: 1 = Address parity error detected 0 = No address parity error
	17	Response hard fail	Hardware failure detected on response
	18	Response parity	Parity error detected on response
	19	PIC and FSB data parity	Data Parity detected on either PIC or FSB access
	20	Processor Signature = 00000F04H: Invalid PIC request All other processors: Reserved	Processor Signature = 00000F04H. Indicates error due to an invalid PIC request access was made to PIC space with WB memory): 1 = Invalid PIC request error 0 = No Invalid PIC request error Reserved
	21	Pad state machine	The state machine that tracks P and N data-strobe relative timing has become unsynchronized or a glitch has been detected.
	22	Pad strobe glitch	Data strobe glitch
	23	Pad address glitch	Address strobe glitch
Other Information	56:24	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

Table 16-10 provides information on interpreting additional family 0FH, model specific fields for cache hierarchy errors. These errors are reported in one of the IA32_MCi_STATUS MSRs. These errors are reported, architecturally, as compound errors with a general form of *0000 0001 RRRR TTLL* in the MCA error code field. See Chapter 15 for how to interpret the compound error code.

16.13.1 Model-Specific Machine Check Error Codes for Intel Xeon Processor MP 7100 Series

Intel Xeon processor MP 7100 series has 5 register banks which contains information related to Machine Check Errors. MCi_STATUS[63:0] refers to all 5 register banks. MC0_STATUS[63:0] through MC3_STATUS[63:0] is the same as on previous generation of Intel Xeon processors within Family 0FH. MC4_STATUS[63:0] is the main error logging for the processor's L3 and front side bus errors. It supports the L3 Errors, Bus and Interconnect Errors Compound Error Codes in the MCA Error Code Field.

Table 16-41. MCi_STATUS Register Bit Definition

Bit Field Name	Bits	Description
MCA_Error_Code	15:0	Specifies the machine check architecture defined error code for the machine check error condition detected. The machine check architecture defined error codes are guaranteed to be the same for all Intel Architecture processors that implement the machine check architecture. See tables below
Model_Specific_Error_Code	31:16	Specifies the model specific error code that uniquely identifies the machine check error condition detected. The model specific error codes may differ among Intel Architecture processors for the same Machine Check Error condition. See tables below
Other_Info	56:32	The functions of the bits in this field are implementation specific and are not part of the machine check architecture. Software that is intended to be portable among Intel Architecture processors should not rely on the values in this field.
PCC	57	Processor Context Corrupt flag indicates that the state of the processor might have been corrupted by the error condition detected and that reliable restarting of the processor may not be possible. When clear, this flag indicates that the error did not affect the processor's state. This bit will always be set for MC errors which are not corrected.
ADDRV	58	MC_ADDR register valid flag indicates that the MC_ADDR register contains the address where the error occurred. When clear, this flag indicates that the MC_ADDR register does not contain the address where the error occurred. The MC_ADDR register should not be read if the ADDRV bit is clear.
MISCV	59	MC_MISC register valid flag indicates that the MC_MISC register contains additional information regarding the error. When clear, this flag indicates that the MC_MISC register does not contain additional information regarding the error. MC_MISC should not be read if the MISCV bit is not set.
EN	60	Error enabled flag indicates that reporting of the machine check exception for this error was enabled by the associated flag bit of the MC_CTL register. Note that correctable errors do not have associated enable bits in the MC_CTL register so the EN bit should be clear when a correctable error is logged.

Table 16-41. MCi_STATUS Register Bit Definition (Contd.)

Bit Field Name	Bits	Description
UC	61	Error uncorrected flag indicates that the processor did not correct the error condition. When clear, this flag indicates that the processor was able to correct the event condition.
OVER	62	Machine check overflow flag indicates that a machine check error occurred while the results of a previous error were still in the register bank (i.e., the VAL bit was already set in the MC_STATUS register). The processor sets the OVER flag and software is responsible for clearing it. Enabled errors are written over disabled errors, and uncorrected errors are written over corrected events. Uncorrected errors are not written over previous valid uncorrected errors.
VAL	63	MC_STATUS register valid flag indicates that the information within the MC_STATUS register is valid. When this flag is set, the processor follows the rules given for the OVER flag in the MC_STATUS register when overwriting previously valid entries. The processor sets the VAL flag and software is responsible for clearing it.

16.13.1.1 Processor Machine Check Status Register MCA Error Code Definition

Intel Xeon processor MP 7100 series use compound MCA Error Codes for logging its CBC internal machine check errors, L3 Errors, and Bus/Interconnect Errors. It defines additional Machine Check error types (IA32_MC4_STATUS[15:0]) beyond those defined in Chapter 15. Table 16-42 lists these model-specific MCA error codes. Error code details are specified in MC4_STATUS [31:16] (see Section 16.13.3), the "Model Specific Error Code" field. The information in the "Other_Info" field (MC4_STATUS[56:32]) is common to the three processor error types and contains a correctable event count and specifies the MC4_MISC register format.

Table 16-42. Incremental MCA Error Code for Intel Xeon Processor MP 7100

Processor MCA_Error_Code (MC4_STATUS[15:0])			
Type	**Error Code**	**Binary Encoding**	**Meaning**
C	Internal Error	0000 0100 0000 0000	Internal Error Type Code
A	L3 Tag Error	0000 0001 0000 1011	L3 Tag Error Type Code
B	Bus and Interconnect Error	0000 100x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations
		0000 101x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations
		0000 110x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations
		0000 1110 0000 1111	Bus and Interconnection Error Type Code
		0000 1111 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations

The **Bold faced** binary encodings are the only encodings used by the processor for MC4_STATUS[15:0].

16.13.2 Other_Info Field (all MCA Error Types)

The MC4_STATUS[56:32] field is common to the processor's three MCA error types (A, B & C).

Table 16-43. Other Information Field Bit Definition

Bit Field Name	Bits	Description
39:32	8-bit Correctable Event Count	Holds a count of the number of correctable events since cold reset. This is a saturating counter; the counter begins at 1 (with the first error) and saturates at a count of 255.
41:40	MC4_MISC format type	The value in this field specifies the format of information in the MC4_MISC register. Currently, only two values are defined. Valid only when MISCV is asserted.
43:42	–	Reserved
51:44	ECC syndrome	ECC syndrome value for a correctable ECC event when the "Valid ECC syndrome" bit is asserted
52	Valid ECC syndrome	Set when correctable ECC event supplies the ECC syndrome
54:53	Threshold-Based Error Status	00: No tracking - No hardware status tracking is provided for the structure reporting this event. 01: Green - Status tracking is provided for the structure posting the event; the current status is green (below threshold). 10: Yellow - Status tracking is provided for the structure posting the event; the current status is yellow (above threshold). 11: Reserved for future use Valid only if Valid bit (bit 63) is set Undefined if the UC bit (bit 61) is set
56:55	–	Reserved

16.13.3 Processor Model Specific Error Code Field

16.13.3.1 MCA Error Type A: L3 Error

Note: The Model Specific Error Code field in MC4_STATUS (bits 31:16).

Table 16-44. Type A: L3 Error Codes

Bit Num	Sub-Field Name	Description	Legal Value(s)
18:16	L3 Error Code	Describes the L3 error encountered	000 - No error 001 - More than one way reporting a correctable event 010 - More than one way reporting an uncorrectable error 011 - More than one way reporting a tag hit 100 - No error 101 - One way reporting a correctable event 110 - One way reporting an uncorrectable error 111 - One or more ways reporting a correctable event while one or more ways are reporting an uncorrectable error
20:19	–	Reserved	00
31:21	–	Fixed pattern	0010_0000_000

16.13.3.2 Processor Model Specific Error Code Field Type B: Bus and Interconnect Error

Note: The Model Specific Error Code field in MC4_STATUS (bits 31:16).

Table 16-45. Type B Bus and Interconnect Error Codes

Bit Num	Sub-Field Name	Description
16	FSB Request Parity	Parity error detected during FSB request phase
17	Core0 Addr Parity	Parity error detected on Core 0 request's address field
18	Core1 Addr Parity	Parity error detected on Core 1 request's address field
19		Reserved
20	FSB Response Parity	Parity error on FSB response field detected
21	FSB Data Parity	FSB data parity error on inbound data detected
22	Core0 Data Parity	Data parity error on data received from Core 0 detected
23	Core1 Data Parity	Data parity error on data received from Core 1 detected
24	IDS Parity	Detected an Enhanced Defer parity error (phase A or phase B)
25	FSB Inbound Data ECC	Data ECC event to error on inbound data (correctable or uncorrectable)
26	FSB Data Glitch	Pad logic detected a data strobe 'glitch' (or sequencing error)
27	FSB Address Glitch	Pad logic detected a request strobe 'glitch' (or sequencing error)
31:28	---	Reserved

Exactly one of the bits defined in the preceding table will be set for a Bus and Interconnect Error. The Data ECC can be correctable or uncorrectable (the MC4_STATUS.UC bit, of course, distinguishes between correctable and uncorrectable cases with the Other_Info field possibly providing the ECC Syndrome for correctable errors). All other errors for this processor MCA Error Type are uncorrectable.

16.13.3.3 Processor Model Specific Error Code Field Type C: Cache Bus Controller Error

Table 16-46. Type C Cache Bus Controller Error Codes

MC4_STATUS[31:16] (MSCE) Value	Error Description
0000_0000_0000_0001 0001H	Inclusion Error from Core 0
0000_0000_0000_0010 0002H	Inclusion Error from Core 1
0000_0000_0000_0011 0003H	Write Exclusive Error from Core 0
0000_0000_0000_0100 0004H	Write Exclusive Error from Core 1
0000_0000_0000_0101 0005H	Inclusion Error from FSB
0000_0000_0000_0110 0006H	SNP Stall Error from FSB
0000_0000_0000_0111 0007H	Write Stall Error from FSB
0000_0000_0000_1000 0008H	FSB Arb Timeout Error
0000_0000_0000_1001 0009H	CBC OOD Queue Underflow/overflow
0000_0001_0000_0000 0100H	Enhanced Intel SpeedStep Technology TM1-TM2 Error
0000_0010_0000_0000 0200H	Internal Timeout error
0000_0011_0000_0000 0300H	Internal Timeout Error
0000_0100_0000_0000 0400H	Intel® Cache Safe Technology Queue Full Error or Disabled-ways-in-a-set overflow

Table 16-46. Type C Cache Bus Controller Error Codes (Contd.)

MC4_STATUS[31:16] (MSCE) Value	Error Description
1100_0000_0000_0001 C001H	Correctable ECC event on outgoing FSB data
1100_0000_0000_0010 C002H	Correctable ECC event on outgoing Core 0 data
1100_0000_0000_0100 C004H	Correctable ECC event on outgoing Core 1 data
1110_0000_0000_0001 E001H	Uncorrectable ECC error on outgoing FSB data
1110_0000_0000_0010 E002H	Uncorrectable ECC error on outgoing Core 0 data
1110_0000_0000_0100 E004H	Uncorrectable ECC error on outgoing Core 1 data
— all other encodings —	Reserved

All errors - except for the correctable ECC types - in this table are uncorrectable. The correctable ECC events may supply the ECC syndrome in the Other_Info field of the MC4_STATUS MSR.

Table 16-47. Decoding Family 0FH Machine Check Codes for Cache Hierarchy Errors

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	15:0		
Model specific error codes	17:16	Tag Error Code	Contains the tag error code for this machine check error: 00 = No error detected 01 = Parity error on tag miss with a clean line 10 = Parity error/multiple tag match on tag hit 11 = Parity error/multiple tag match on tag miss
	19:18	Data Error Code	Contains the data error code for this machine check error: 00 = No error detected 01 = Single bit error 10 = Double bit error on a clean line 11 = Double bit error on a modified line
	20	L3 Error	This bit is set if the machine check error originated in the L3 it can be ignored for invalid PIC request errors): 1 = L3 error 0 = L2 error
	21	Invalid PIC Request	Indicates error due to invalid PIC request access was made to PIC space with WB memory): 1 = Invalid PIC request error 0 = No invalid PIC request error
	31:22	Reserved	Reserved
Other Information	39:32	8-bit Error Count	Holds a count of the number of errors since reset. The counter begins at 0 for the first error and saturates at a count of 255.
	56:40	Reserved	Reserved
Status register validity indicators[1]	63:57		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 15, "Machine-Check Architecture," for more information.

17. Updates to Chapter 17, Volume 3B

Change bars and green text show changes to Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

Changes to this chapter: Update to Section 17.4.1, "IA32_DEBUGCTL MSR". Update to Table 17-4, "LBR Stack Size and TOS Pointer Range".

CHAPTER 17
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 18 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 17.17, "Time-Stamp Counter".
- Features which allow monitoring of shared platform resources such as the L3 cache are described in Section 17.18, "Intel® Resource Director Technology (Intel® RDT) Monitoring Features".
- Features which enable control over shared platform resources are described in Section 17.19, "Intel® Resource Director Technology (Intel® RDT) Allocation Features".

17.1 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.
- **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.

17.2 DEBUG REGISTERS

Eight debug registers (see Figure 17-1 for 32-bit operation and Figure 17-2 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.

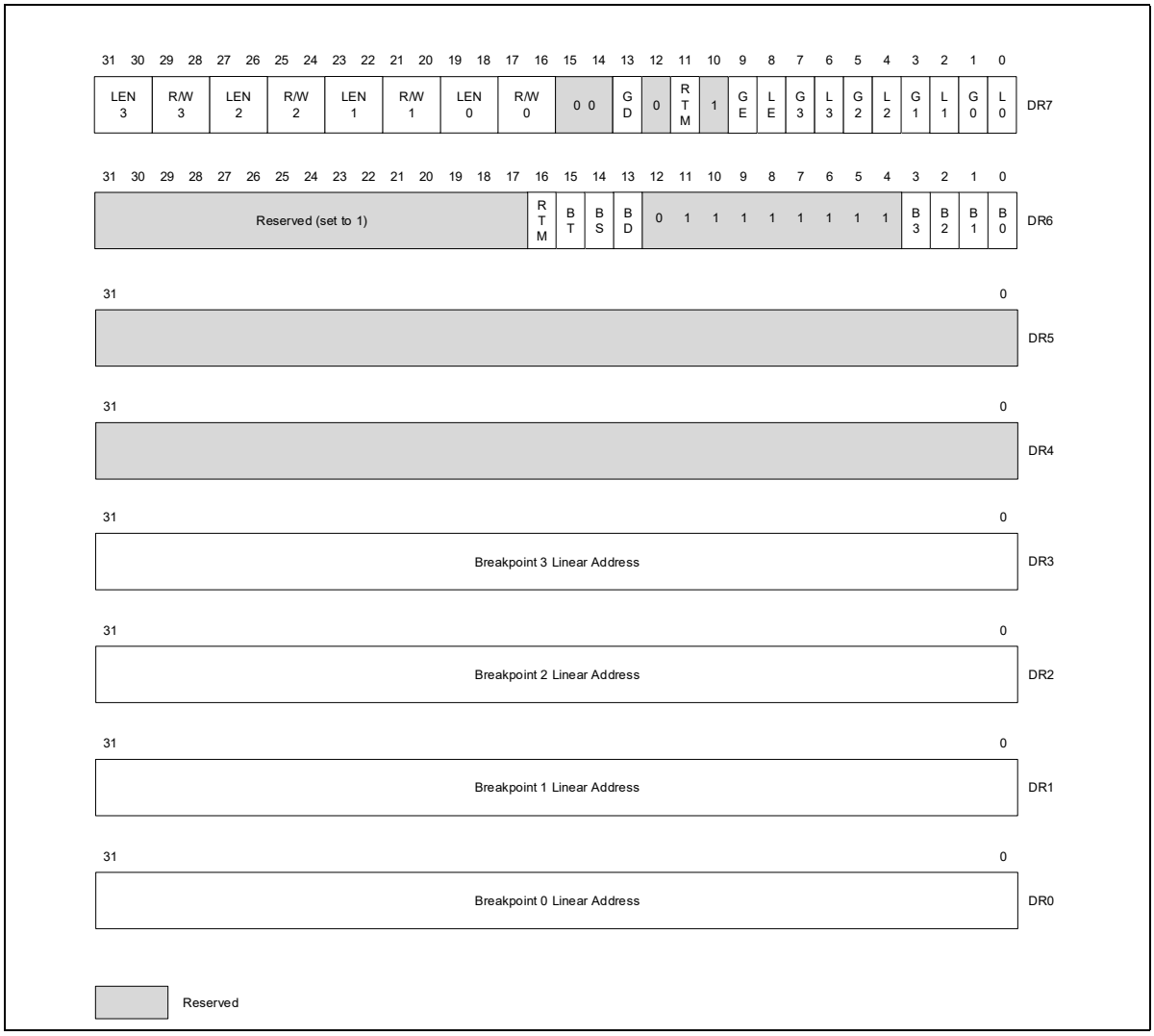


Figure 17-1. Debug Registers

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 though 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 17.2.4).
- 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.

17.2.1 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 17-1). Breakpoint comparisons are made before physical address translation occurs. The contents of debug register DR7 further specifies breakpoint conditions.

17.2.2 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.

17.2.3 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 17-1). 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.
- **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 17.2.4, "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 7.2.1, "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 17.3.3). 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.

17.2.4 Debug Control Register (DR7)

The debug control register (DR7) enables or disables breakpoints and sets breakpoint conditions (see Figure 17-1). 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 17.3.3). 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 RW*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 17.2.5, "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 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.

17.2.5 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 17-1 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 17-1. Breakpoint Examples

Debug Register Setup			
Debug Register	**R/W*n***	**Breakpoint Address**	**LEN*n***
DR0 DR1 DR2 DR3	R/W0 = 11 (Read/Write) R/W1 = 01 (Write) R/W2 = 11 (Read/Write) R/W3 = 01 (Write)	A0001H A0002H B0002H C0000H	LEN0 = 00 (1 byte) LEN1 = 00 (1 byte) LEN2 = 01) (2 bytes) LEN3 = 11 (4 bytes)
Data Accesses			
Operation		**Address**	**Access Length (In Bytes)**
Data operations that trap - Read or write - Read or write - Write - Write - Read or write - Read or write - Read or write - Write - Write - Write		 A0001H A0001H A0002H A0002H B0001H B0002H B0002H C0000H C0001H C0003H	 1 2 1 2 4 1 2 4 2 1
Data operations that do not trap - Read or write - Read - Read or write - Read or write - Read - Read or write		 A0000H A0002H A0003H B0000H C0000H C0004H	 1 1 4 2 2 4

17.2.6 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 17-2). 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). Break point conditions for 8-byte memory read/writes are supported in all modes.

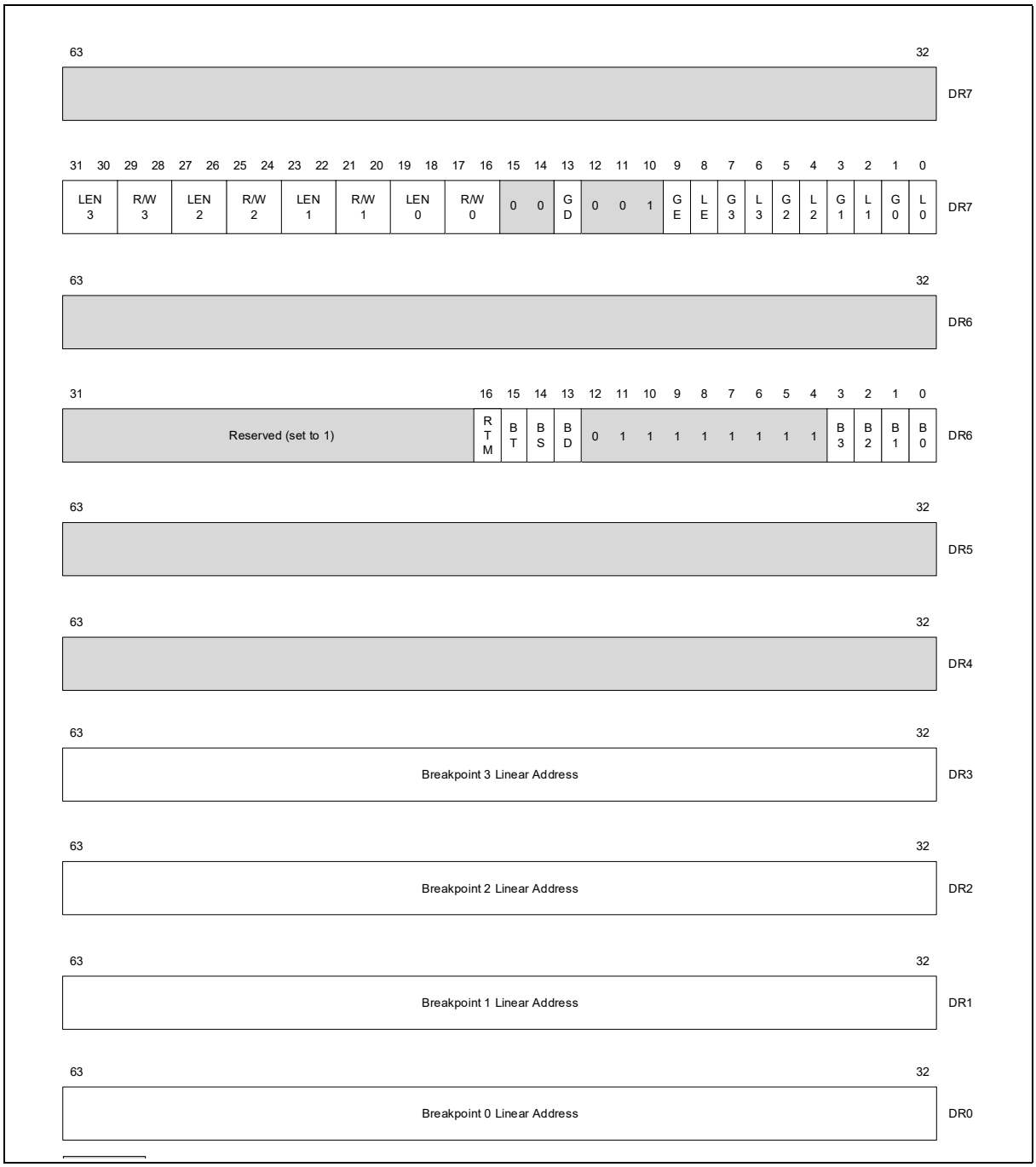


Figure 17-2. DR6/DR7 Layout on Processors Supporting Intel® 64 Architecture

17.3 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.

17.3.1 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 17-2 shows the states of these flags following the generation of each kind of breakpoint condition.

Instruction-breakpoint and general-detect condition (see Section 17.3.1.3, "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 6, "Interrupt 1—Debug Exception (#DB)," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Table 17-2. 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/Wn = 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/Wn = 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/Wn = 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/Wn = 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

17.3.1.1 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 6.8.3, "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.

17.3.1.2 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 *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.

17.3.1.3 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.

17.3.1.4 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.

17.3.1.5 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.

17.3.2 Breakpoint Exception (#BP)—Interrupt Vector 3

The breakpoint exception (interrupt 3) is caused by execution of an INT3 instruction. See Chapter 6, "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.

17.3.3 Debug Exceptions, Breakpoint Exceptions, and Restricted Transactional Memory (RTM)

Chapter 16, "Programming with Intel® Transactional Synchronization Extensions," 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]).

17.4 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 17.5, "Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors)"
- Section 17.6, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Microarchitecture"
- Section 17.9, "Last Branch, Interrupt, and Exception Recording for Processors based on Nehalem Microarchitecture"

— Section 17.10, "Last Branch, Interrupt, and Exception Recording for Processors based on Sandy Bridge Microarchitecture"

— Section 17.11, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Haswell Microarchitecture"

— Section 17.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture"

— Section 17.14, "Last Branch, Interrupt, and Exception Recording (Intel® Core™ Solo and Intel® Core™ Duo Processors)"

— Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)"

— Section 17.16, "Last Branch, Interrupt, and Exception Recording (P6 Family Processors)"

The following subsections of Section 17.4 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).

17.4.1 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 17-3 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 17.5.1, "LBR Stack" (Intel® Core™2 Duo and Intel Atom® processor family) and Section 17.9.1, "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 17.4.3, "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 17.4.4, "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 17.4.9, "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 17.4.5, "Branch Trace Store (BTS)," for a description of this mechanism.

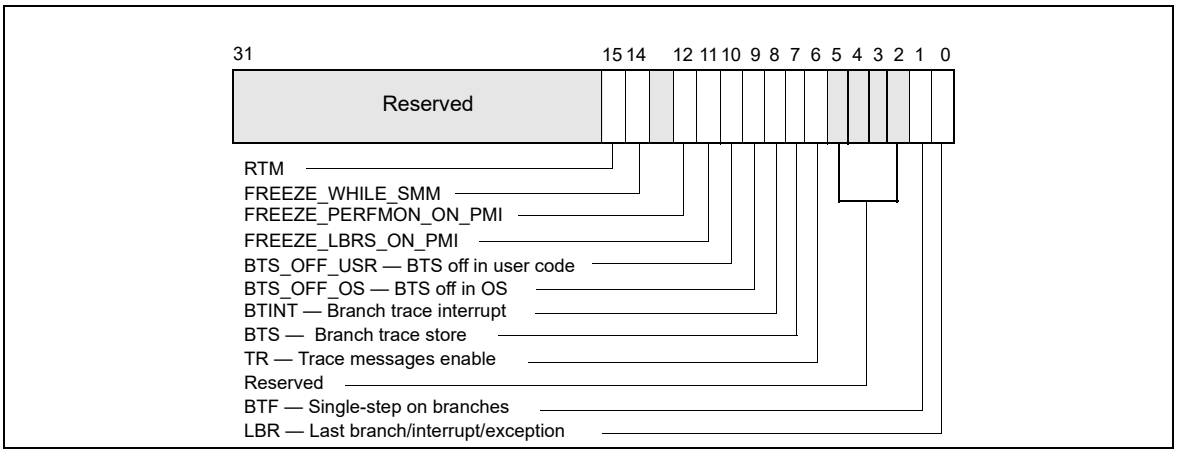


Figure 17-3. IA32_DEBUGCTL MSR for Processors based on Intel Core microarchitecture

- **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 17.13.2.
- **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 17.13.2.
- **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 17.4.7 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 17.4.7 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 14.6.11 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 19.8 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 17.3.3.

17.4.2 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.

17.4.3 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.

17.4.4 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 17.4.2, "Monitoring Branches, Exceptions, and Interrupts."

For P6 processor family, Pentium M processor family, 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).

17.4.4.1 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.

17.4.5 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 17.4.9.2, "Setting Up the DS Save Area." for additional details.

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 7.2.1, "Task-State Segment (TSS)."

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.

17.4.6 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[bit 4] = 1.

The CPL-qualified branch trace mechanism is described in Section 17.4.9.4. 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.

17.4.7 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.ArchPerfMonVerID[bits 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.ArchPerfMonVerID[bits 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 19-3). The PMCs affected include both general-purpose counters and fixed-function counters (see Section 19.6.2.1, “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 19-1) or IA32_FIXED_CTR_CTRL MSR (see Figure 19-2).

- The PEBS buffer is almost full and reaches the interrupt threshold.
- The BTS buffer is almost full and reaches the interrupt threshold.

Table 17-3 compares the interaction of the processor with the PMI handler using the legacy versus streamlined Freeza_Perfmon_On_PMI interface.

Table 17-3. 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

17.4.8 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 17-4 lists the LBR stack size and TOS pointer range for several processor families according to the CPUID signatures of DisplayFamily_DisplayModel encoding (see CPUID instruction in Chapter 3 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*).

Table 17-4. 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 17.12.

The last branch recording mechanism tracks not only branch instructions (like JMP, Jcc, LOOP and CALL instructions), but also other operations that cause a change in the instruction pointer (like 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 17-4) that store source and destination address of recent branches (see Figure 17-3):
  - 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 17-4.

17.4.8.1 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.

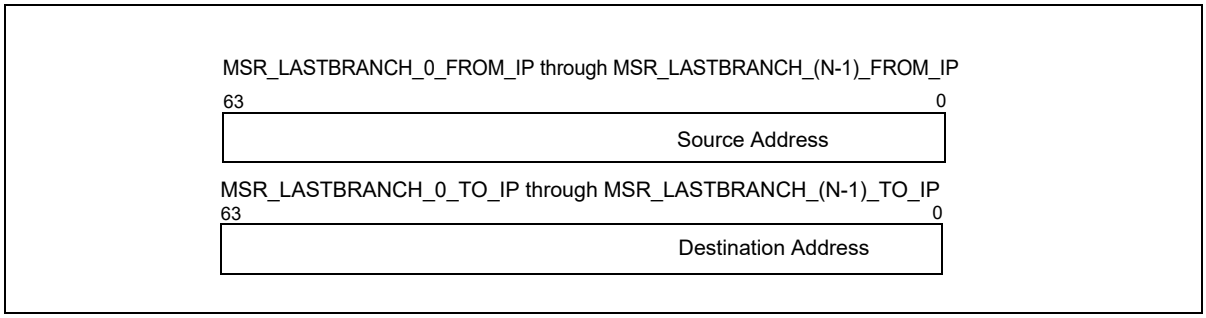


Figure 17-4. 64-bit Address Layout of LBR MSR

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, 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, 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 17.6).

— **000111B (64-bit LIP record format), Flags, 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_CAPAB_MSR] (bit 15).

17.4.8.2 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.

17.4.8.3 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.

17.4.9 BTS and DS Save Area

The **Debug store (DS)** feature flag (bit 21), returned by CPUID.1: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 17.4.5, "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 19.6.2.4, "Processor Event Based Sampling (PEBS)," and the relevant PEBS sub-sections across the core PMU sections in Chapter 19, "Performance Monitoring."

When CPUID.1: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 19.6.2.4.2 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 19.5.3.1 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 17-5). 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.

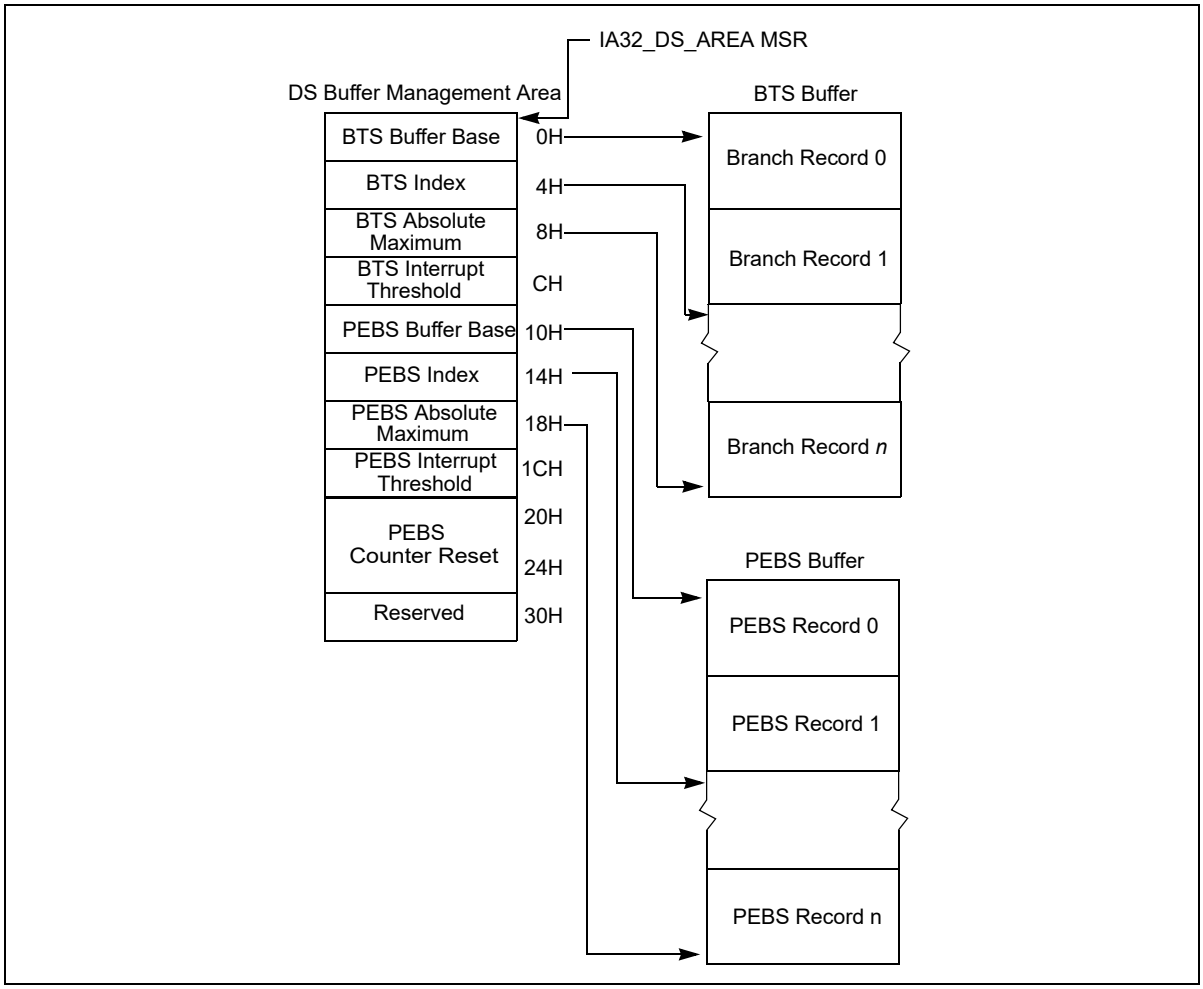


Figure 17-5. DS Save Area Example[1]

NOTES:

1. This example represents the format for a system that supports PEBS on only one counter.

Figure 17-6 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).

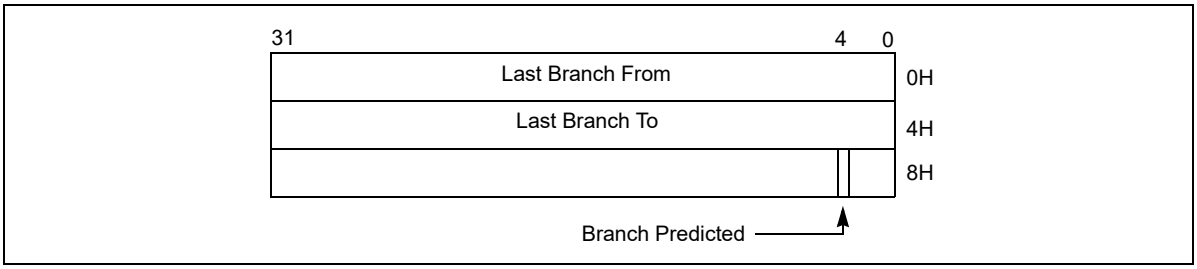


Figure 17-6. 32-bit Branch Trace Record Format

Figure 17-7 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.

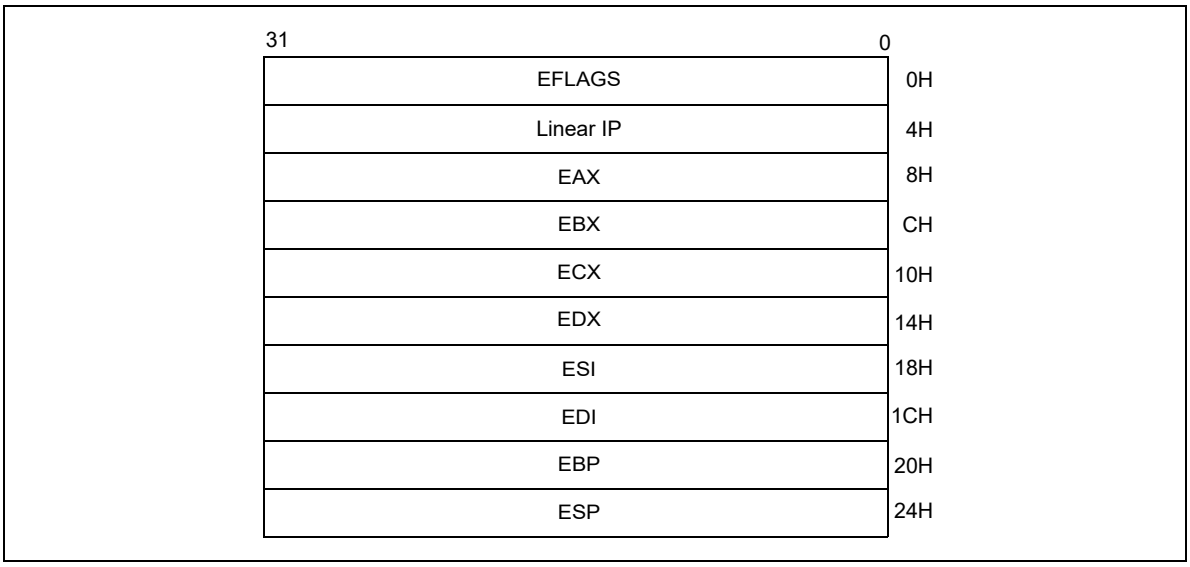


Figure 17-7. PEBS Record Format

17.4.9.1 64 Bit Format of the DS Save Area

When DTES64 = 1 (CPUID.1.ECX[2] = 1), the structure of the DS save area is shown in Figure 17-8.

When DTES64 = 0 (CPUID.1.ECX[2] = 0) and IA-32e mode is active, the structure of the DS save area is shown in Figure 17-8. If IA-32e mode is not active the structure of the DS save area is as shown in Figure 17-5.

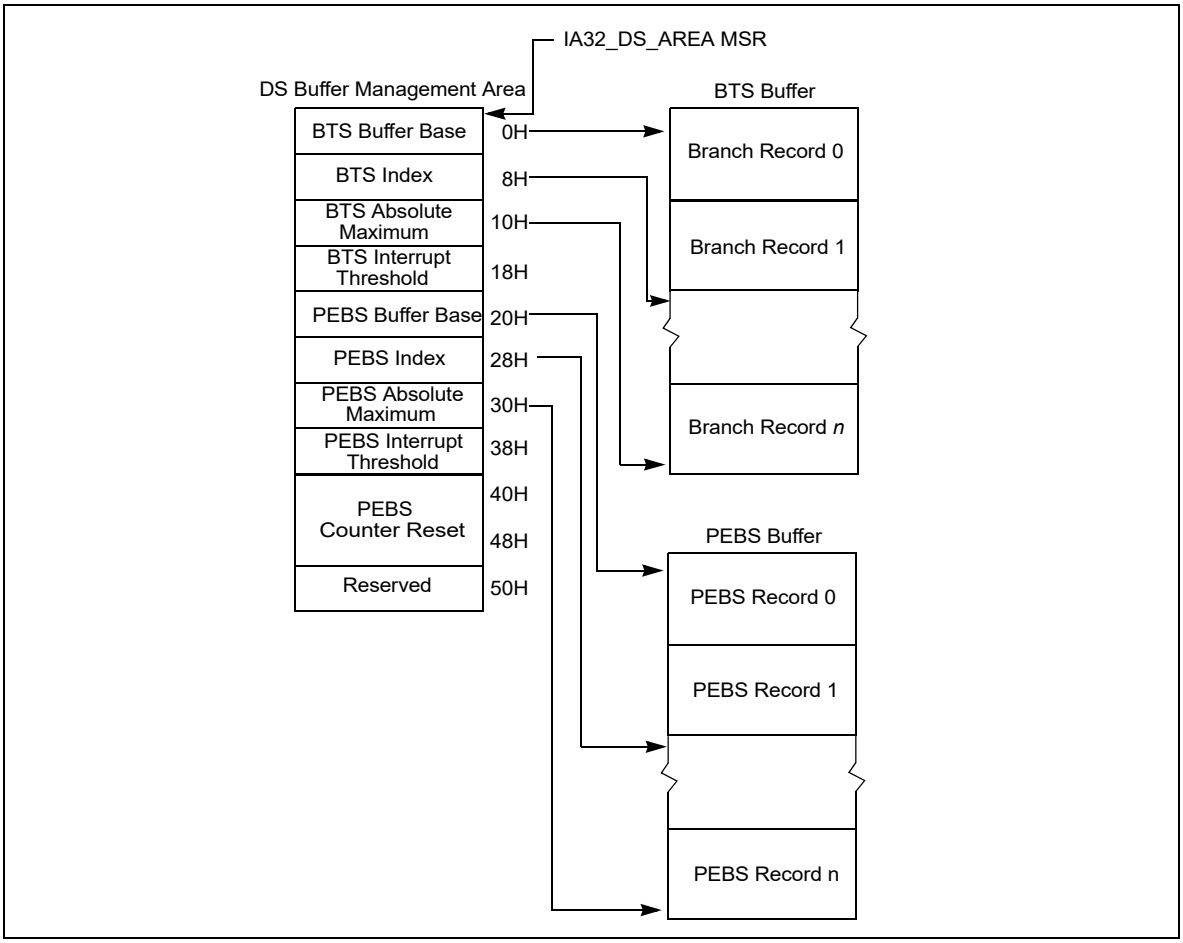


Figure 17-8. IA-32e Mode DS Save Area Example[1]

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 17-6, but each field is 8 bytes in length. This makes each BTS record 24 bytes (see Figure 17-9). The structure of a PEBS record is similar to that shown in Figure 17-7, 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 17-10).

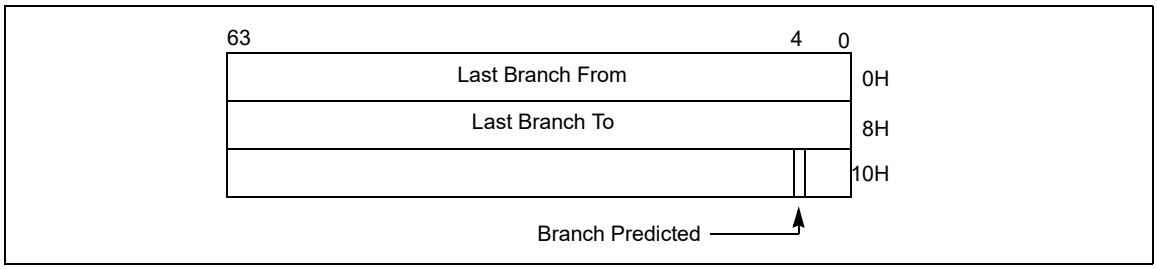


Figure 17-9. 64-bit Branch Trace Record Format

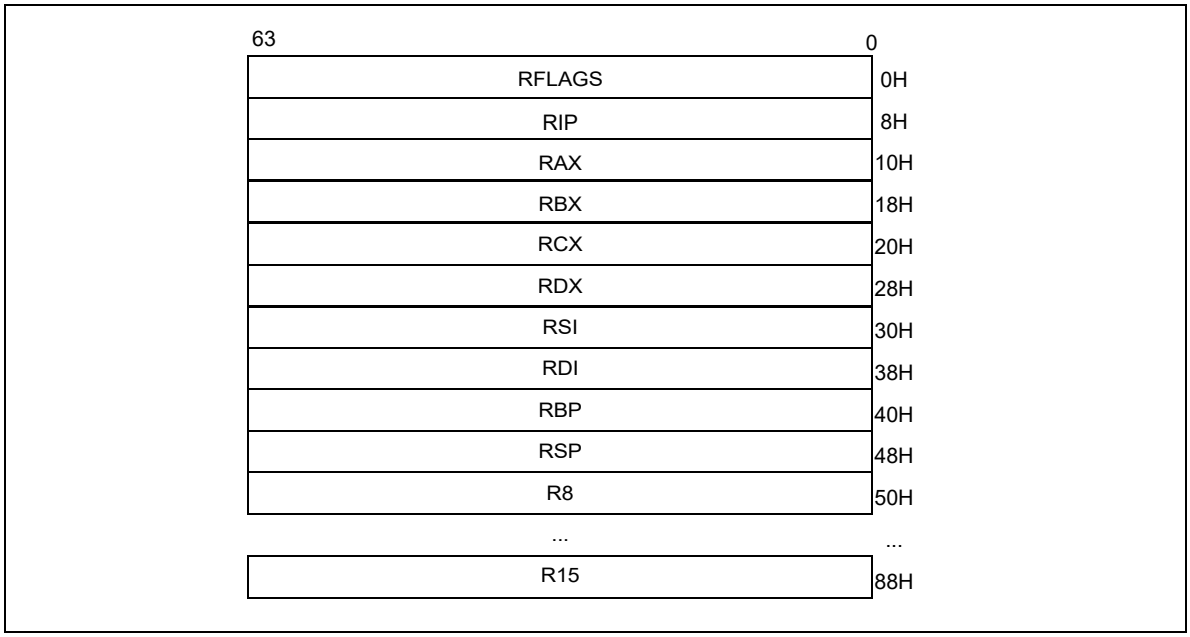


Figure 17-10. 64-bit PEBS Record Format

Fields in the buffer management area of a DS save area are described in Section 17.4.9.

The format of a branch trace record and a PEBS record are the same as the 64-bit record formats shown in Figures 17-9 and Figures 17-10, 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 17.4.9.3 and Section 17.4.9.4.

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 19-3).

17.4.9.2 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 19.6.2.4.1, "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 17.4.9, "BTS and DS Save Area," and Section 17.4.9.1, "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 10.5.1, "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 17.4.9.5, "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 19.3.9.1 and Section 19.9.5 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 used only on systems with an enabled APIC. The LVT Performance Counter entry in the APCI must be initialized to use an interrupt gate instead of the trap gate.

17.4.9.3 Setting Up the BTS Buffer

Three flags in the MSR_DEBUGCTLA MSR (see Table 17-5), IA32_DEBUGCTL (see Figure 17-3), or MSR_DEBUGCTLB (see Figure 17-16) 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 17-5. 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.

17.4.9.4 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 17-6.

Table 17-6. 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

17.4.9.5 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.

17.5 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 17.4.1 for a description of the flags. See Figure 17-3 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 17.5.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts**
  - See Section 17.4.2 and Section 17.4.3. 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 17.4.4.
- **Last exception records —** See Section 17.13.3.
- **Branch trace store and CPL-qualified BTS —** See Section 17.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11) —** see Section 17.4.7 for legacy Freeze_LBRs_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12) —** see Section 17.4.7 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 17.4.1.

17.5.1 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 17.4.8.1. 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.

17.5.2 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 17-11.

17.6 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 17.5.2 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 17-13.
- Elapsed cycle information is added to MSR_LASTBRANCH_x_TO_IP. Format is shown in Table 17-7.
- Misprediction info is reported in the upper bits of MSR_LASTBRANCH_x_FROM_IP. MISPRED bit format is shown in Table 17-8.
- Streamlined Freeze_LBRs_On_PMI operation; see Section 17.12.2.
- LBR MSRs may be cleared when MWAIT is used to request a C-state that is numerically higher than C1; see Section 17.12.3.

Table 17-7. 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 17.4.8.1 for address format.
Cycle Count (Saturating)	63:48	R/W	Elapsed core clocks since last update to the LBR stack.

17.7 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 17.6 with the following changes:

- Enumeration of new LBR format: encoding 00111b in IA32_PERF_CAPABILITIES[5:0] is supported, see Section 17.4.8.1.
- Each LBR stack entry consists of three MSRs:
  - MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 17-9.
  - MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 17-9.
  - MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data. Layout is the same as Table 17-16.

17.8 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 17-11.

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 17.4.8.1.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.

17.9 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 17.4.1 for a description of the flags. See Figure 17-11 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 17.9.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts —** See Section 17.4.2 and Section 17.4.3. 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 17.4.4. However, not all BTM messages are observable using the Intel® QPI link.
- **Last exception records —** See Section 17.13.3.
- **Branch trace store and CPL-qualified BTS —** See Section 17.4.6 and Section 17.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11) —** see Section 17.4.7 for legacy Freeze_LBRs_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12) —** see Section 17.4.7 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 17.4.1.

Processors based on Nehalem microarchitecture provide additional capabilities:

- **Independent control of uncore PMI —** The IA32_DEBUGCTL MSR provides a bit field (see Figure 17-11) 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.

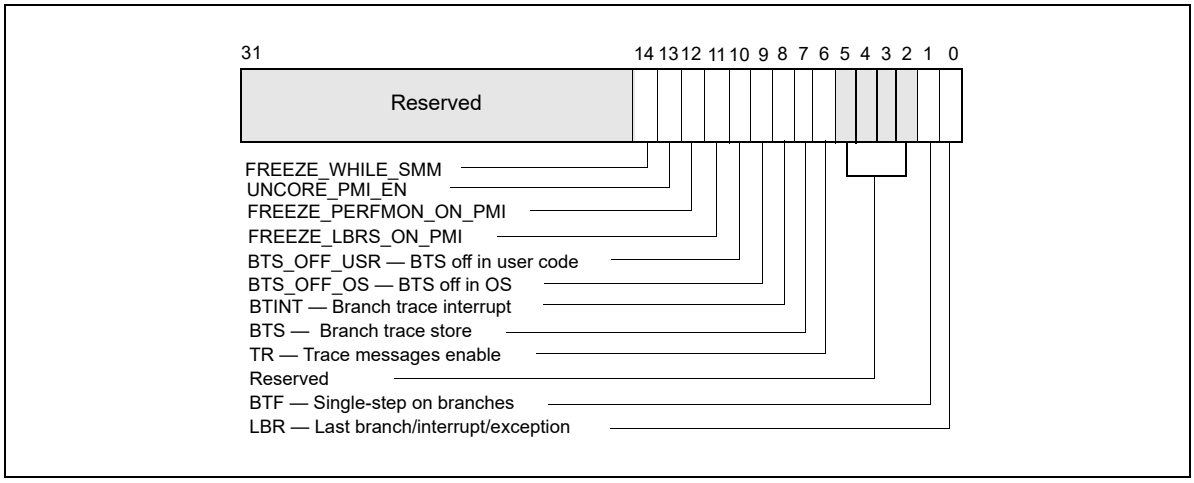


Figure 17-11. IA32_DEBUGCTL MSR for Processors Based on Nehalem Microarchitecture

17.9.1 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 17-8 and Table 17-9.

Table 17-8. MSR_LASTBRANCH_x_FROM_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the "branch from" address. See Section 17.4.8.1 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 17-9. MSR_LASTBRANCH_x_TO_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the "branch to" address. See Section 17.4.8.1 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 17-10.

Table 17-10. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_1AH	16	0 to 15

17.9.2 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 17-11.

Table 17-11. 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

17.10 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 17.9, "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 17-12.

Table 17-12. 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

17.11 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 17.10, "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 17-13. If MSR_LBR_SELECT.EN_CALLSTACK is clear, the LBR facility will capture branches normally as described in Section 17.10.

Table 17-13. 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.)

17.11.1 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 17-14 and Table 17-9.

Table 17-14. 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 17.4.8.1 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.

17.12 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 17.4.8.1.
- Each LBR stack entry consists of a triplets of MSRs:

  - MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 17-9.
  - MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 17-9.
  - 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 supports the same LBR filtering capabilities as described in Table 17-13.

Table 17-15. 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

17.12.1 MSR_LBR_INFO_x MSR

The layout of each MSR_LBR_INFO_x MSR is shown in Table 17-16.

Table 17-16. 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.

17.12.2 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 17.4.7).

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.

17.12.3 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.

17.13 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 consist 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.1:EDX.DS[bit 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.1:ECX.DS-CPL[bit 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 17-17, Figure 17-12, and Section 17.13.2, "LBR Stack for Processors Based on Intel NetBurst® Microarchitecture."
- **Last exception record** — See Section 17.13.3, "Last Exception Records."

17.13.1 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 17-12 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 17.13.2, "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 17.4.3, "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 17.4.4, "Branch Trace Messages."

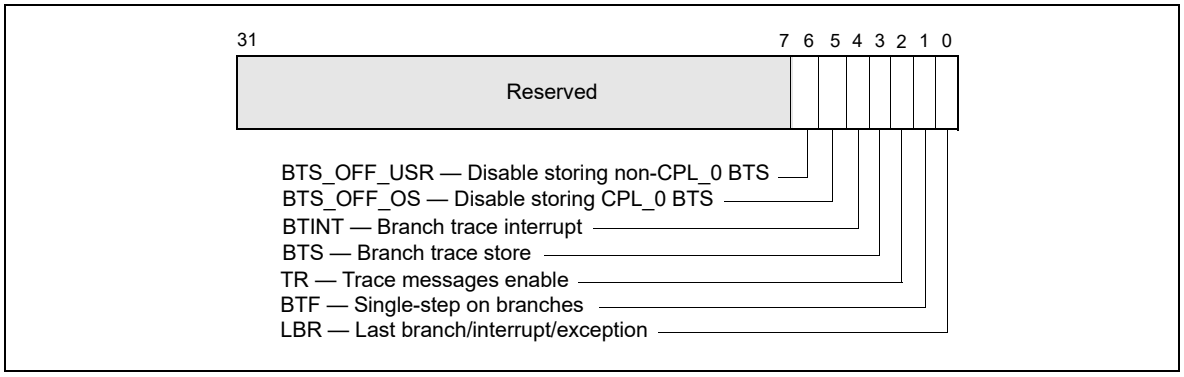


Figure 17-12. MSR_DEBUGCTLA MSR for Pentium 4 and Intel Xeon Processors

- **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 17.4.9, "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 17.4.5, "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 17.13.2, "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 17.13.2, "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 17-12. 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.

17.13.2 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 it maximum value, it wraps around to 0. See Table 17-17 and Figure 17-12.

Table 17-17. 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 17-13 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.

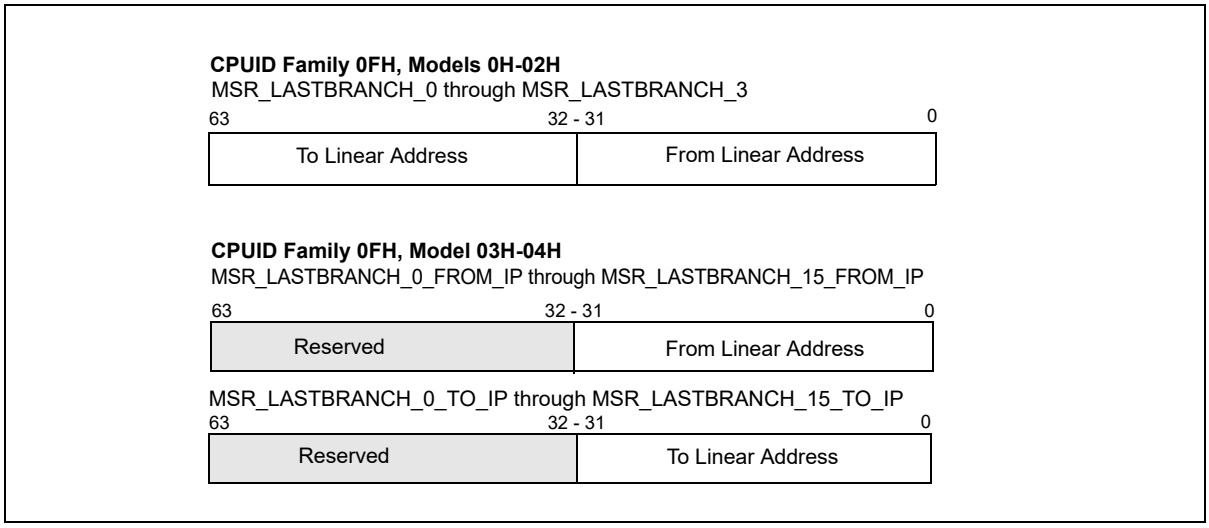


Figure 17-13. LBR MSR Branch Record Layout for the Pentium 4 and Intel® Xeon® Processor Family

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.

17.13.3 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.

17.14 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 17-14 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 17.4.3, "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 17.4.4, "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 17.4.9, "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 17.4.5, "Branch Trace Store (BTS)," for a description of this mechanism.

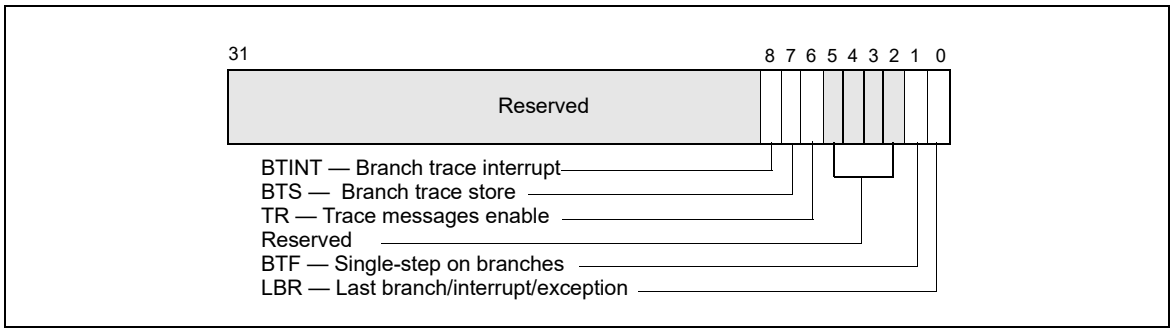


Figure 17-14. IA32_DEBUGCTL MSR for Intel® Core™ Solo and Intel® Core™ Duo Processors

- **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 17.4.5, "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 17-15.

- **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 Last-ExceptionFromIP MSRs found in P6 family processors.

For details, see Section 17.12, "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*.

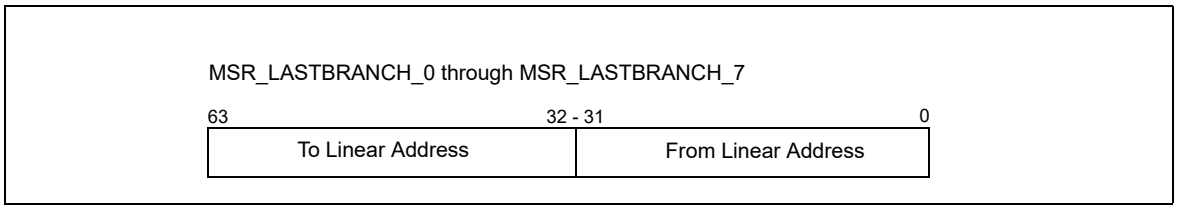


Figure 17-15. LBR Branch Record Layout for the Intel® Core™ Solo and Intel® Core™ Duo Processor

17.15 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 17-16 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 17.4.3, "Single-Stepping on Branches," for more information about the BTF flag.
  - **PB*i* (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 BP*i*# pin when a breakpoint match occurs. When a PB*i* 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 17.4.4, "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 17.4.9, "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 17.4.5, "Branch Trace Store (BTS)," for a description of this mechanism.

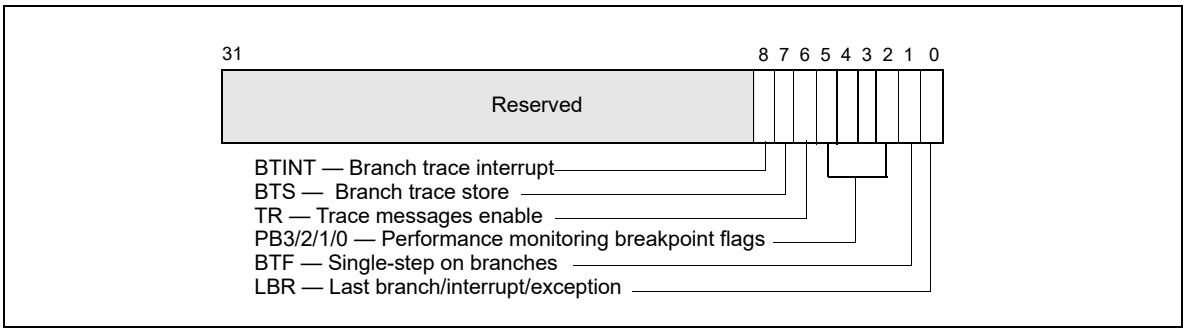


Figure 17-16. MSR_DEBUGCTLB MSR for Pentium M Processors

- **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 17.4.5, "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 17-17.
- **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.

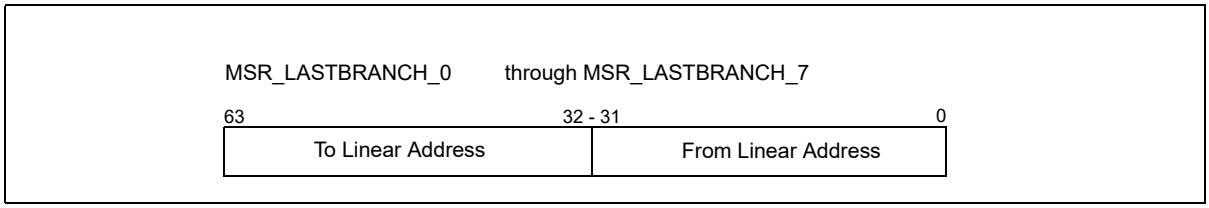


Figure 17-17. LBR Branch Record Layout for the Pentium M Processor

For more detail on these capabilities, see Section 17.13.3, "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*.

17.16 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: DEBUGCTLMSR, 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.

17.16.1 DEBUGCTLMSR Register

The version of the DEBUGCTLMSR 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 17-18 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.

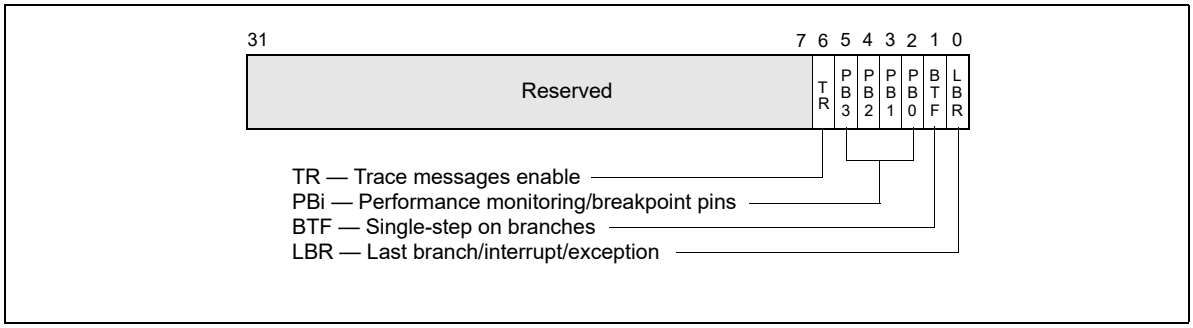


Figure 17-18. DEBUGCTLMSR Register (P6 Family Processors)

- **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 17.4.3, "Single-Stepping on Branches."
- **PB*i* (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 BP*i*# pin when a breakpoint match occurs. When a PB*i* 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 17.4.4, "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.

17.16.2 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.

17.16.3 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.

17.17 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.1:EDX.TSC[bit 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 19.7.2 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 19.6.4.5, "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.

17.17.1 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.

17.17.2 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.

17.17.3 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 than 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.(EAX=07H, ECX=0H):EBX.TSC_ADJUST (bit 1).

17.17.4 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 leaf 15H expresses the frequency relationship between the ART hardware and TSC.

If CPUID.15H:EBX[31:0] != 0 and CPUID.80000007H:EDX[InvariantTSC] = 1, the following linearity relationship holds between TSC and the ART hardware:

TSC_Value = (ART_Value * CPUID.15H:EBX[31:0])/ CPUID.15H:EAX[31:0] + K

Where 'K' is an offset that can be adjusted by a privileged agent[1].

When ART hardware is reset, both invariant TSC and K are also reset.

17.18 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.

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

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).

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.

17.18.1 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 ID**s (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. Before reading back monitored data software must configure 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).

17.18.2 Enabling Monitoring: Usage Flow

Figure 17-19 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.

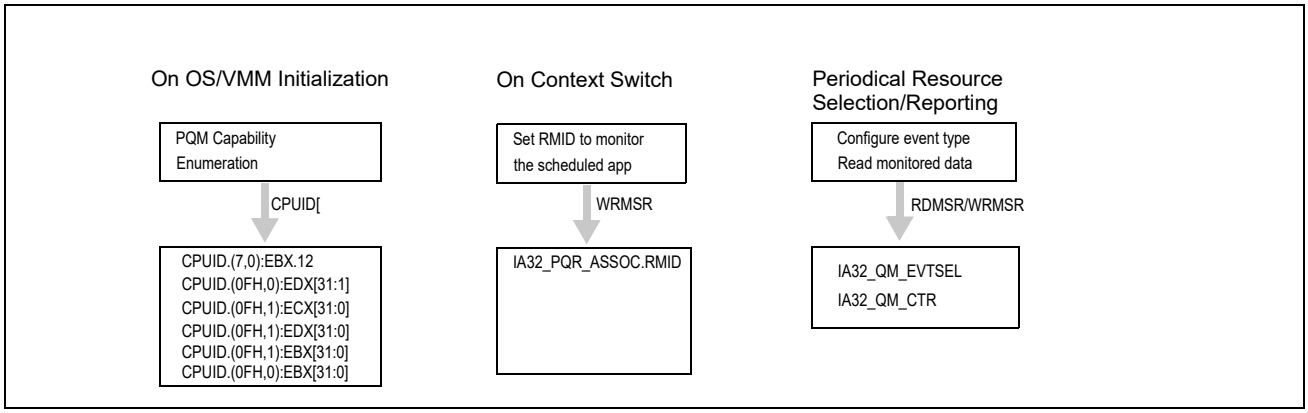


Figure 17-19. Platform Shared Resource Monitoring Usage Flow

17.18.3 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.(EAX=07H, ECX=0):EBX.PQM[bit 12] reports 1, the processor provides the following programming interfaces for shared resource monitoring, including Cache Monitoring Technology:

- CPUID leaf function 0FH (Shared Resource Monitoring Enumeration leaf) provides information on available resource types (see Section 17.18.4), and monitoring capabilities for each resource type (see Section 17.18.5). 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 17.18.6.
- 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 17.18.7.
- 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. Execute CPUID with EAX=0 to discover the "cpuid_maxLeaf" supported in the processor;
2. If cpuid_maxLeaf >= 7, then execute CPUID with EAX=7, ECX= 0 to verify CPUID.(EAX=07H, ECX=0):EBX.PQM[bit 12] is set;
3. If CPUID.(EAX=07H, ECX=0):EBX.PQM[bit 12] = 1, then execute CPUID with EAX=0FH, ECX= 0 to query available resource types that support monitoring;
4. If CPUID.(EAX=0FH, ECX=0):EDX.L3[bit 1] = 1, then execute CPUID with EAX=0FH, ECX= 1 to query the specific capabilities of L3 Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring.
5. If CPUID.(EAX=0FH, ECX=0):EDX reports additional resource types supporting monitoring, then execute CPUID with EAX=0FH, ECX set to a corresponding resource type ID (ResID) as enumerated by the bit position of CPUID.(EAX=0FH, ECX=0):EDX.

17.18.4 Monitoring Resource Type and Capability Enumeration

CPUID leaf function 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. executing CPUID with EAX=0FH and ECX=0H. In the initial implementation, L3 cache is the only resource type available. Each

supported resource type is represented by a bit in CPUID.(EAX=0FH, ECX=0):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 17-21 and Figure 17-22). Reserved bits of CPUID.(EAX=0FH, ECX=0):EDX[31:2] correspond to unsupported sub-leaves of the CPUID.0FH leaf. Additionally, CPUID.(EAX=0FH, ECX=0H):EBX reports the highest RMID value of any resource type that supports monitoring in the processor.

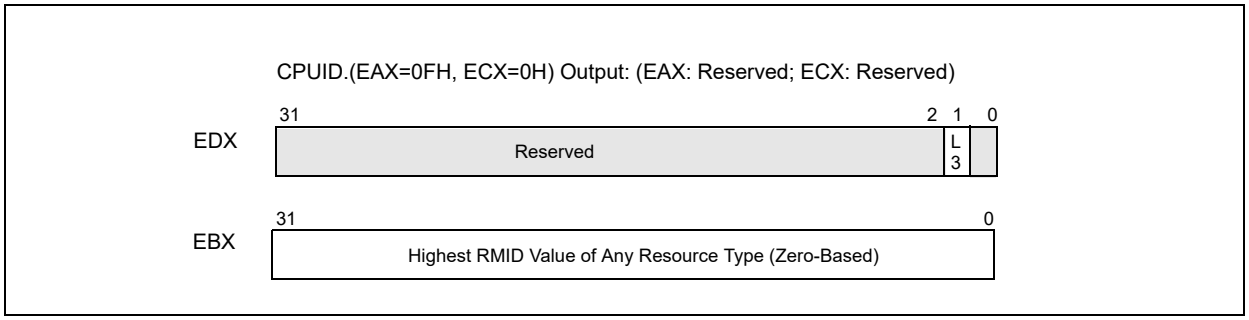


Figure 17-20. CPUID.(EAX=0FH, ECX=0H) Monitoring Resource Type Enumeration

17.18.5 Feature-Specific Enumeration

Each additional sub-leaf of CPUID.(EAX=0FH, ECX=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.

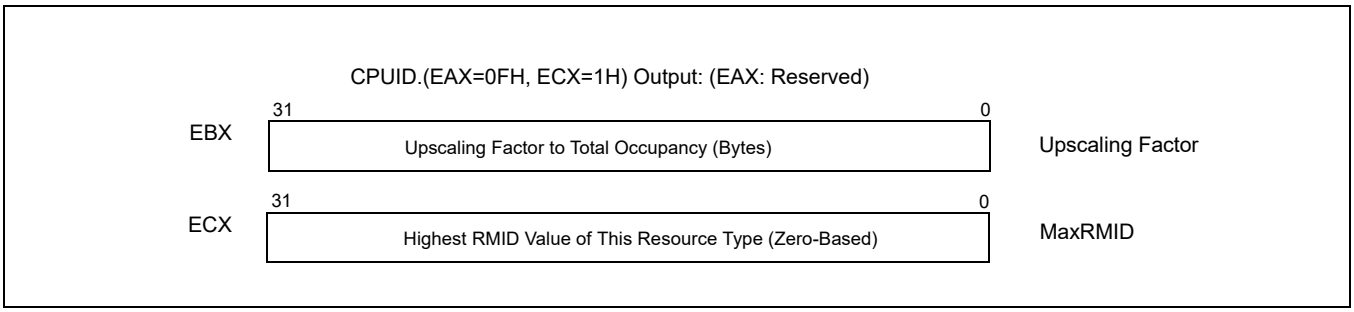


Figure 17-21. L3 Cache Monitoring Capability Enumeration Data (CPUID.(EAX=0FH, ECX=1H))

For each supported Cache Monitoring resource type, hardware supports only a finite number of RMIDs. CPUID.(EAX=0FH, ECX=1H).ECX enumerates the highest RMID value that can be monitored with this resource type, see Figure 17-21.

CPUID.(EAX=0FH, ECX=1H).EDX specifies a bit vector that is used to look up the EventID (See Figure 17-22 and Table 17-18) 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.(EAX=0FH, ECX=1H).EBX, see Figure 17-21.

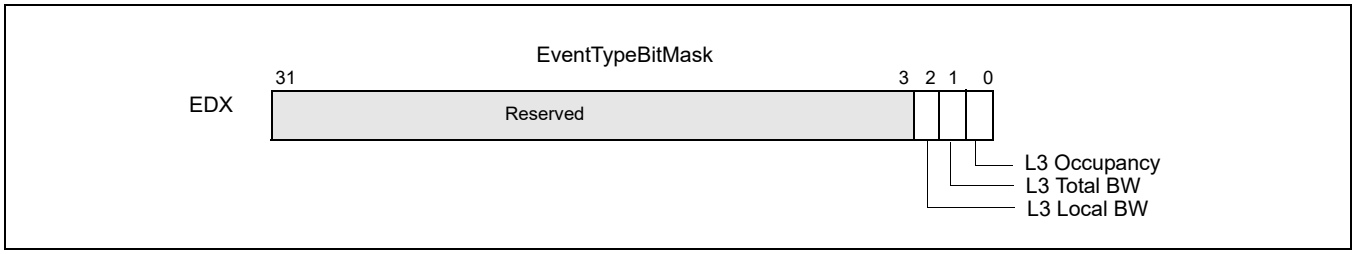


Figure 17-22. L3 Cache Monitoring Capability Enumeration Event Type Bit Vector (CPUID.(EAX=0FH, ECX=1H))

17.18.5.1 Cache Monitoring Technology

On processors for which Cache Monitoring Technology supports the L3 cache occupancy event, CPUID.(EAX=0FH, ECX=1H).EDX would return with only bit 0 set. The corresponding event ID can be looked up from Table 17-18. The L3 occupancy data accumulated in IA32_QM_CTR can be converted to total occupancy (in bytes) by multiplying with CPUID.(EAX=0FH, ECX=1H).EBX.

Event codes for Cache Monitoring Technology are discussed in the next section.

17.18.5.2 Memory Bandwidth Monitoring

On processors that monitoring supports Memory Bandwidth Monitoring using ResID=1 (L3), two additional bits will be set in the vector at CPUID.(EAX=0FH, ECX=1H).EDX:

- CPUID.(EAX=0FH, ECX=1H).EDX[bit 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.(EAX=0FH, ECX=1H).EDX[bit 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).

The corresponding Event ID can be looked up from Table 17-18. The L3 bandwidth data accumulated in IA32_QM_CTR can be converted to total bandwidth (in bytes) using CPUID.(EAX=0FH, ECX=1H).EBX.

Table 17-18. 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

17.18.6 Monitoring Resource RMID Association

After Monitoring and sub-features has 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 17-23. 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 Ceil (LOG_2 (1 + CPUID.(EAX=0FH, ECX=0):EBX[31:0])). The value of IA32_PQR_ASSOC after power-on is 0.

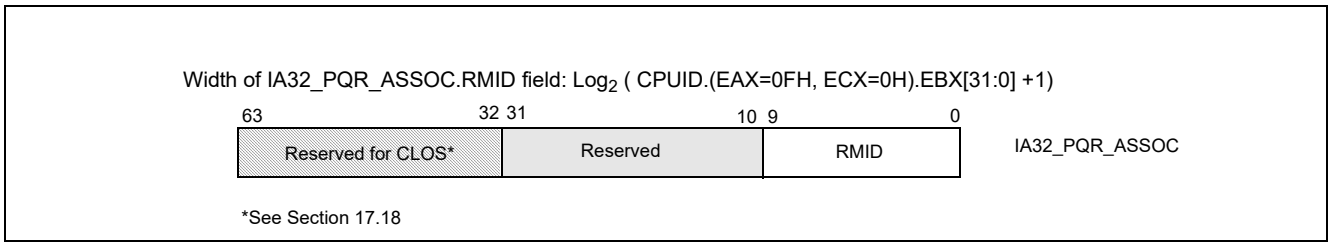


Figure 17-23. IA32_PQR_ASSOC MSR

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.

17.18.7 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 17-24. Bits IA32_QM_EVTSEL.EvtID (bits 7:0) specify 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 17-18. 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 17.18.5, 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 bit 63 and 62 are both clear. 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.(EAX=0FH, ECX=1H).EBX.

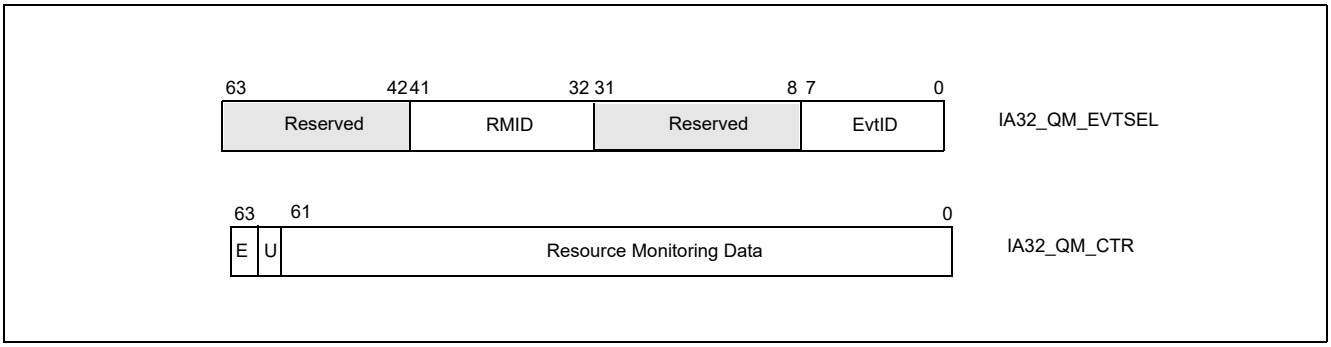


Figure 17-24. IA32_QM_EVTSEL and IA32_QM_CTR MSRs

17.18.8 Monitoring Programming Considerations

Figure 17-23 illustrates how system software can program IA32_QOSEVTSEL and IA32_QM_CTR to perform resource monitoring.

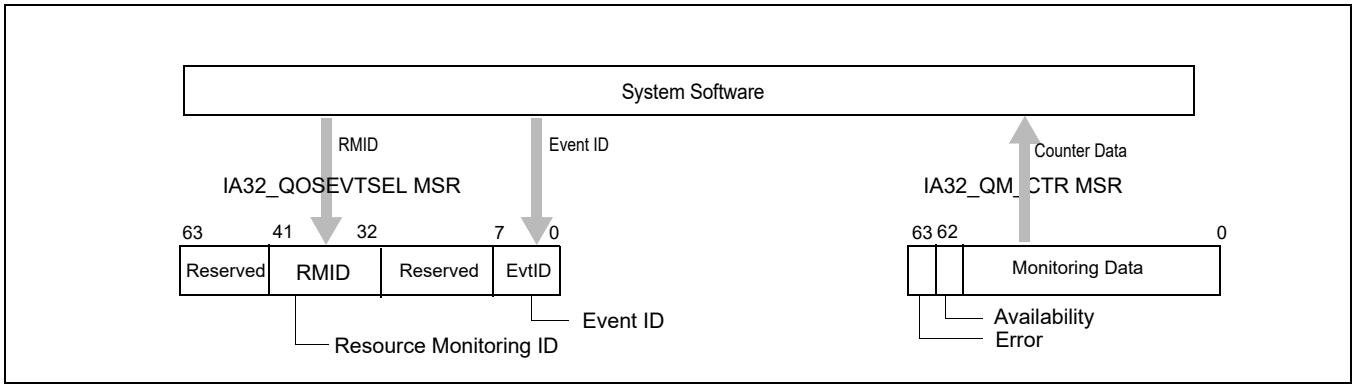


Figure 17-25. Software Usage of Cache Monitoring Resources

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 will be proportional to 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 future changes to counter width will be enumerated to software.

17.18.8.1 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 maxRMID is used.

17.18.8.2 Monitoring Operation With Power Saving Features

Note that 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.

17.18.8.3 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.

17.18.8.4 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.

17.19 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 Processor Scalable 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.

17.19.1 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, and
- 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 17-26. 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 17-26 the L3 Cache is shown as an example resource.

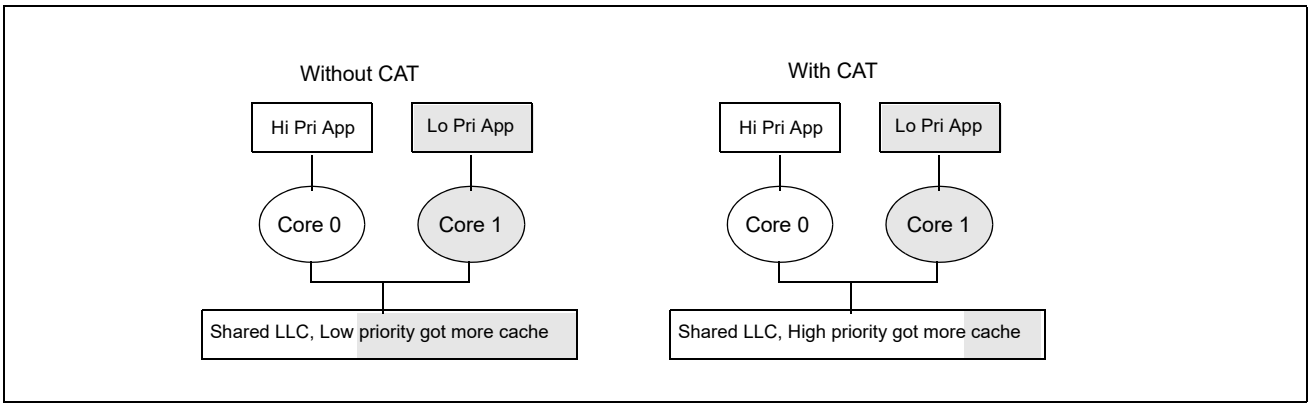


Figure 17-26. Cache Allocation Technology Enables Allocation of More Resources to High Priority Applications

17.19.2 Cache Allocation Technology Architecture

The fundamental goal of Cache Allocation Technology is to enable resource allocation based on application priority or Class of Service (COS or 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 COS when an application, thread or VM is scheduled.

The usage of Classes of Service (COS) are consistent across resources and a COS may have multiple resource control attributes attached, which reduces software overhead at context swap time. Rather than adding new types of COS tags per resource for instance, the COS 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 COS 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 COS to an executing software thread (i.e. associating the active CR3 of a logical processor with the COS in IA32_PQR_ASSOC),
- Implementation-dependent mechanisms to indicate which COS is associated with a memory access and to enforce the cache allocation on a per COS 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 COS may vary for each resource type, and these details can be enumerated via CPUID.

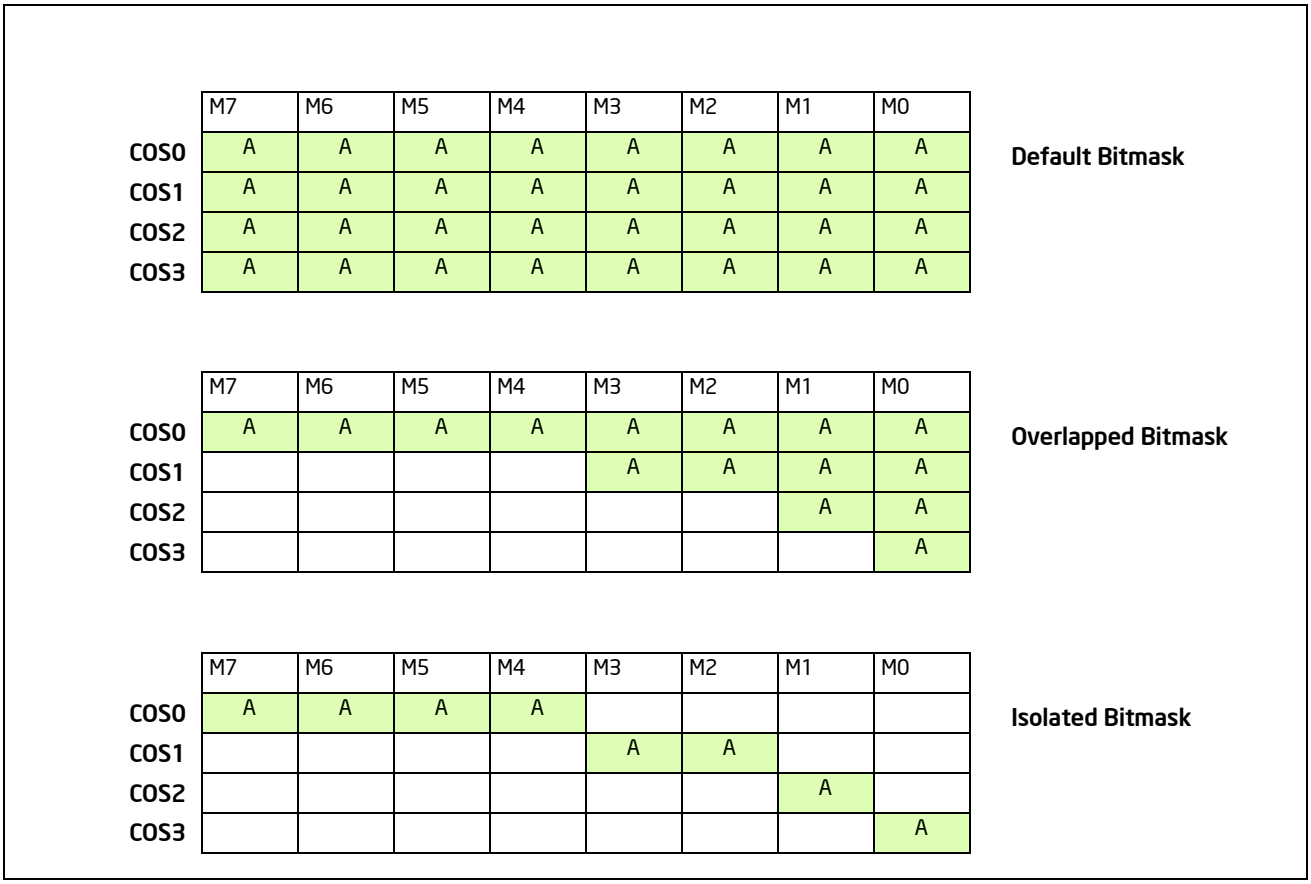


Figure 17-27. Examples of Cache Capacity Bitmasks

Sample cache capacity bitmasks for a bit length of 8 are shown in Figure 17-27. Please note that all (and only) contiguous '1' combinations are allowed (e.g. FFFFH, 0FF0H, 003CH, etc.). 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 17-27 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 COS are implementation-dependent) have full access to the cache. The second case shows an overlapped case, which would allow some lower-priority threads 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 COS0 should typically be considered and configured as the highest priority COS, followed by COS1, and so on, though there is no hardware restriction enforcing this mapping. When the system boots all threads are initialized to COS0, 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.

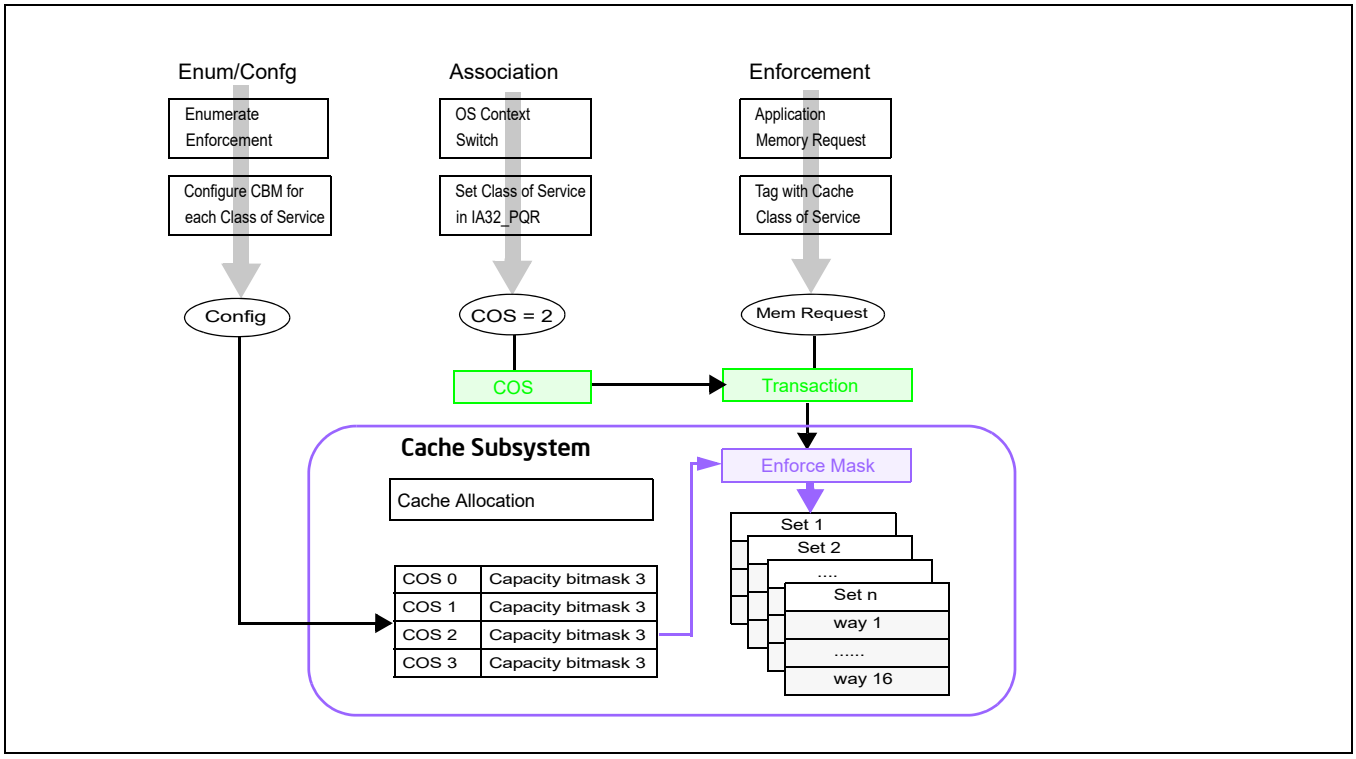


Figure 17-28. Class of Service and Cache Capacity Bitmasks

Figure 17-28 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. 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 COS (i.e. the corresponding COS in the PQR) and all requests to the CAT-capable resource from that logical processor are tagged with that COS (in other words, the application thread is configured to belong to a specific COS). 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.

17.19.3 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 (COS). 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 17.19.6.3 and Section 17.19.7 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 17-29),
- the range of COS for CAT is re-indexed, with the lower-half of the COS 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 17.19.5).

An example of the operating mode of CDP is shown in Figure 17-29. Shown at the top are traditional CAT usage models where capacity masks map 1:1 with a COS number to enable control over the cache space which a given COS (and thus applications, threads or VMs) may occupy. Shown at the bottom are example mask configurations where CDP is enabled, and each COS 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-COS basis, depending on application and system needs.

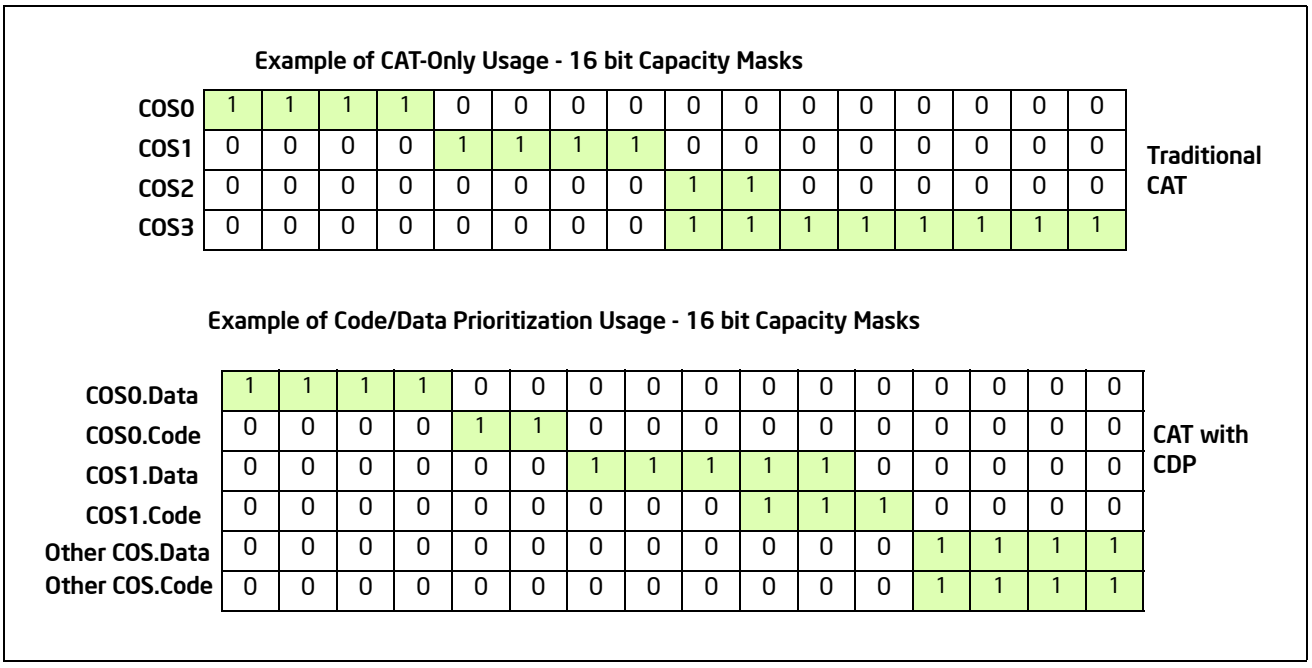


Figure 17-29. Code and Data Capacity Bitmasks of CDP

When CDP is enabled, the existing mask space for CAT-only operation is split. As an example if the system supports 16 CAT-only COS, 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 COS is reduced by half. Code/Data masks are defined per-COS 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.

17.19.4 Enabling Cache Allocation Technology Usage Flow

Figure 17-30 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.

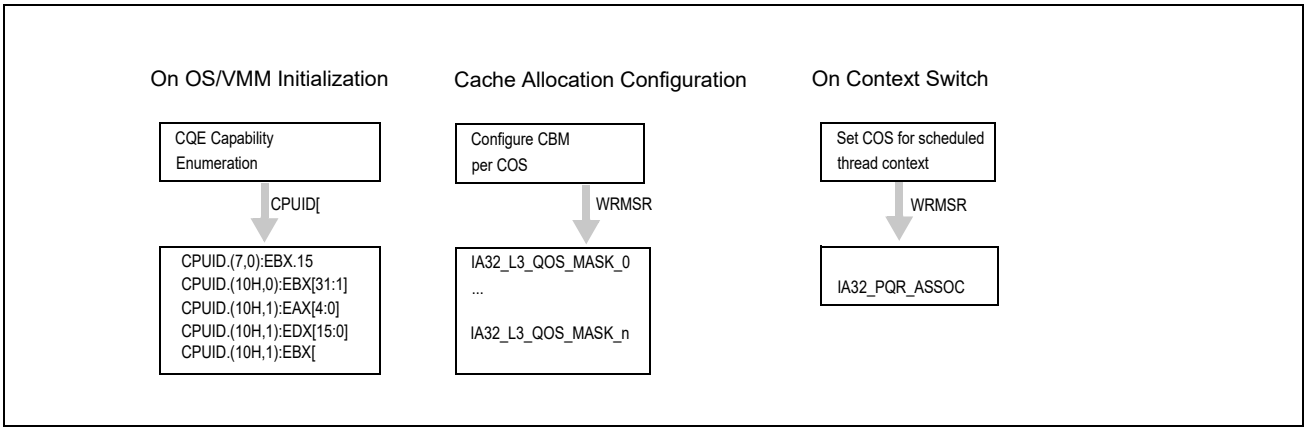


Figure 17-30. Cache Allocation Technology Usage Flow

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.

17.19.4.1 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.(EAX=07H, ECX=0):EBX.PQE[bit 15] reports 1, the processor supports software control over shared processor resources. Software must use CPUID leaf 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 leaf function 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 17.19.4.2).
- 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 COS, i.e. the range between 0 and CPUID.(EAX=10H, ECX=ResID):EDX[15:0], inclusive. See Section 17.19.4.3 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 17.19.4.3 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 15.5.2, “System Software Recommendation for Managing CMCI and Machine Check Resources”. Software may also use CPUID leaf 4 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 COS field that OS/VMM can use to assign a logical processor to an available COS. The set of COS are common across all allocation features, meaning that multiple features may be supported in the same processor without additional software COS management overhead at context swap time. See Section 17.19.4.4 for details.

17.19.4.2 Cache Allocation Technology: Resource Type and Capability Enumeration

CPUID leaf function 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 executing CPUID with EAX=10H and ECX=0H. Each supported resource type is represented by a bit field in

CPUID.(EAX=10H, ECX=0):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 17-31).

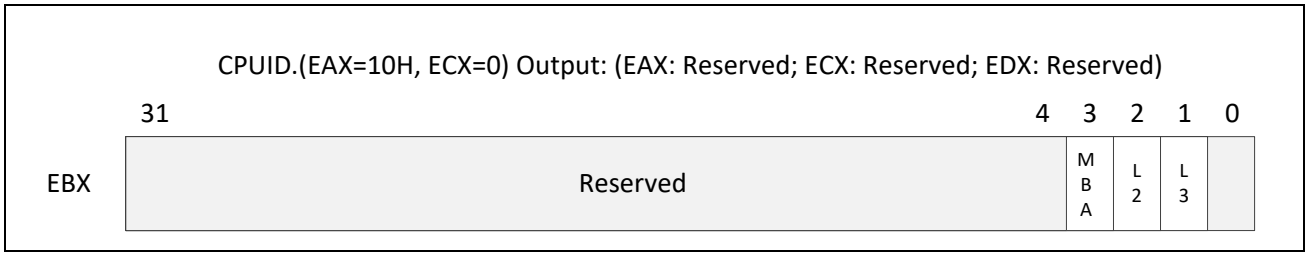


Figure 17-31. CPUID.(EAX=10H, ECX=0H) Available Resource Type Identification

— For ECX>0, EAX[4:0] reports the length of the capacity bitmask length (ECX=1 or 2 for L2 CAT or L3 CAT respectively) using minus-one notation, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.

- Sub-functions of CPUID.EAX=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.(EAX=10H, ECX=0):EBX[31:1] in order to obtain additional feature details.
- CAT capability for L3 is enumerated by CPUID.(EAX=10H, ECX=1H), see Figure 17-32. The specific CAT capabilities reported by CPUID.(EAX=10H, ECX=1) are:

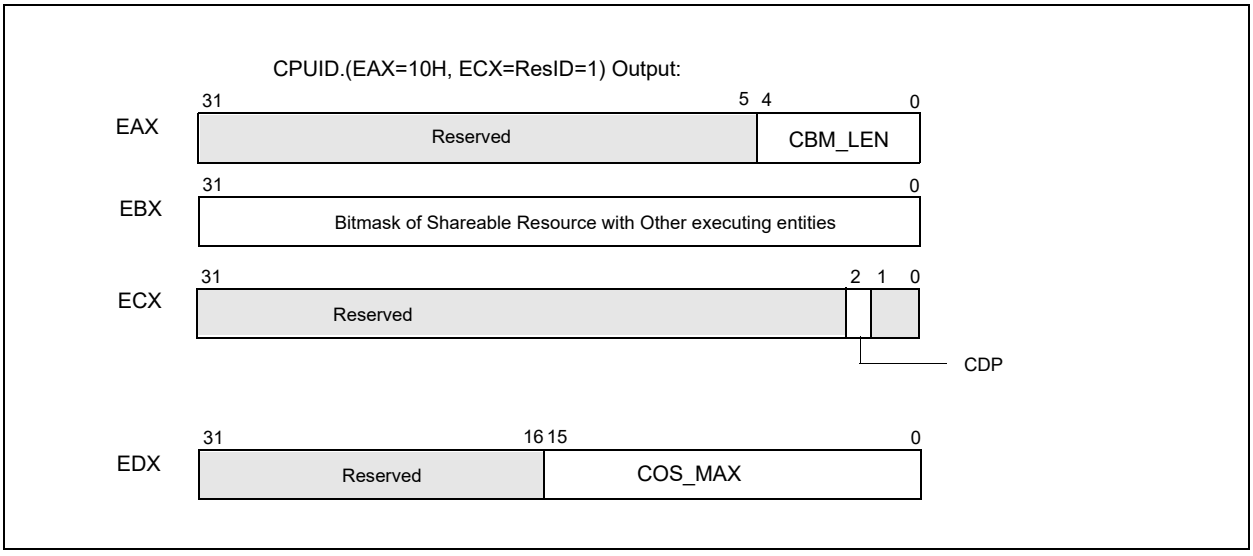


Figure 17-32. L3 Cache Allocation Technology and CDP Enumeration

— CPUID.(EAX=10H, ECX=ResID=1):EAX[4:0] reports the length of the capacity bitmask length using minus-one notation, i.e. a value of 15 corresponds to the capability bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.

— CPUID.(EAX=10H, ECX=1):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.(EAX=10H, ECX=1):ECX.CDP[bit 2]: If 1, indicates L3 Code and Data Prioritization Technology is supported (see Section 17.19.5). Other bits of CPUID.(EAX=10H, ECX=1):ECX are reserved.

— CPUID.(EAX=10H, ECX=1):EDX[15:0] reports the maximum COS supported for the resource (COS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported COS). Bits 31:16 are reserved.

- CAT capability for L2 is enumerated by CPUID.(EAX=10H, ECX=2H), see Figure 17-33. The specific CAT capabilities reported by CPUID.(EAX=10H, ECX=2) are:

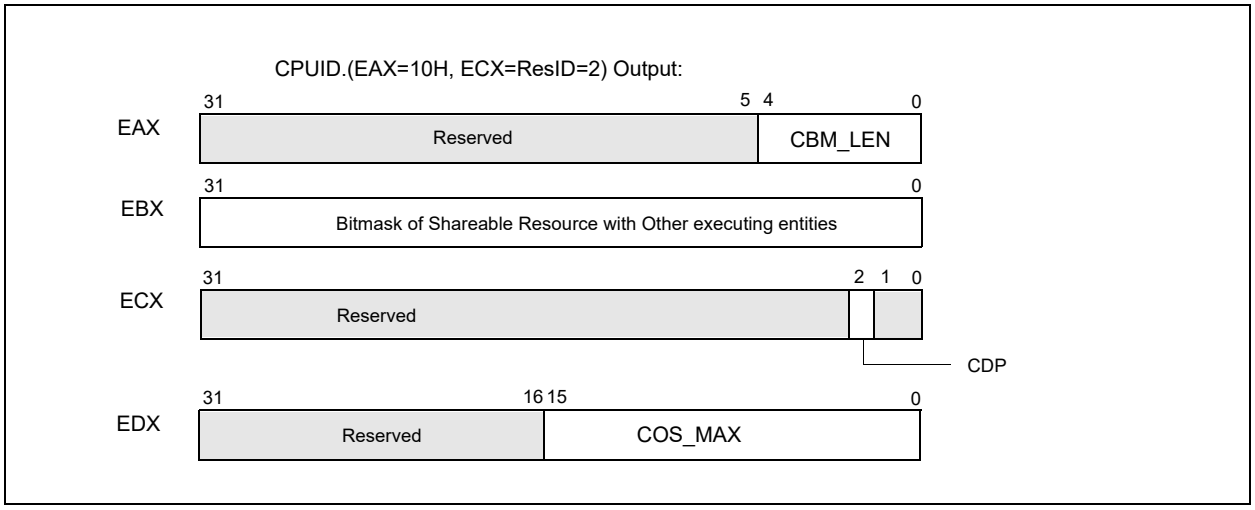


Figure 17-33. L2 Cache Allocation Technology

— CPUID.(EAX=10H, ECX=ResID=2):EAX[4:0] reports the length of the capacity bitmask length using minus-one notation, i.e. a value of 15 corresponds to the capability bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.

— CPUID.(EAX=10H, ECX=2):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.(EAX=10H, ECX=2):ECX.CDP[bit 2]: If 1, indicates L2 Code and Data Prioritization Technology is supported (see Section 17.19.6). Other bits of CPUID.(EAX=10H, ECX=2):ECX are reserved.

— CPUID.(EAX=10H, ECX=2):EDX[15:0] reports the maximum COS supported for the resource (COS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported COS). Bits 31:16 are reserved.

A note on migration of Classes of Service (COS): Software should minimize migrations of COS across logical processors (across threads or cores), as a reduction in the performance of the Cache Allocation Technology feature may result if COS 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 COS migration across processor logical threads and processor cores.

17.19.4.3 Cache Allocation Technology: Cache Mask Configuration

After determining the length of the capacity bitmasks (CBM) and number of COS supported using CPUID (see Section 17.19.4.2), each COS 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 COS, i.e. the range between 0 and CPUID.(EAX=10H, ECX=ResID):EDX[15:0], inclusive, and 'resourceType' corresponds to a specific resource as enumerated by the set bits of CPUID.(EAX=10H, ECX=0):EAX[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 resourceType 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.

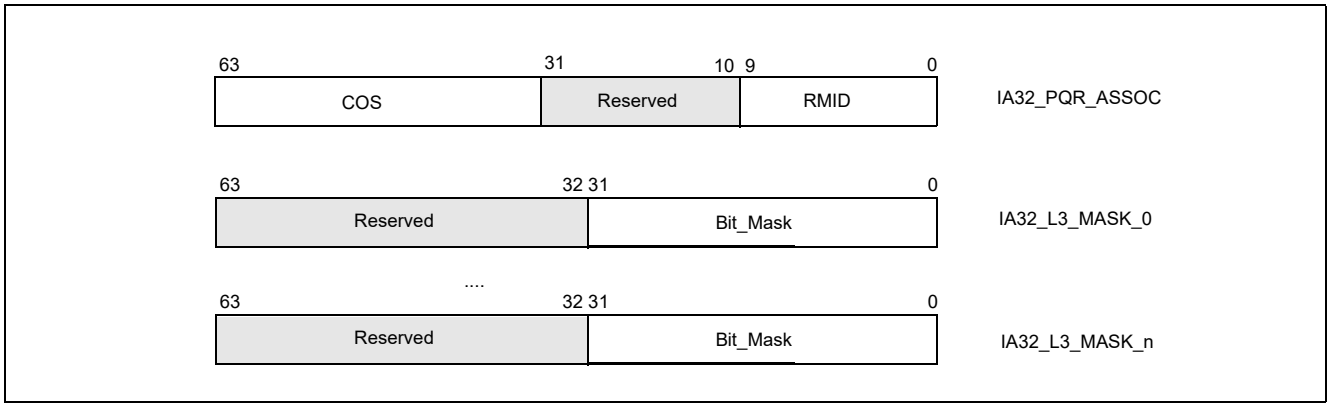


Figure 17-34. IA32_PQR_ASSOC, IA32_L3_MASK_n MSRs

- 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 17-34 and Figure 17-35 provide an overview of the relevant registers.

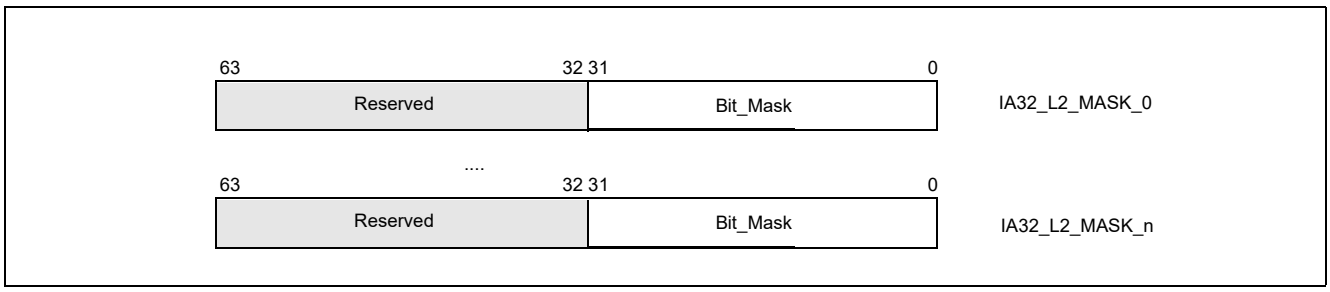


Figure 17-35. IA32_L2_MASK_n MSRs

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 (COS) using the IA32_PQR_ASSOC MSR as described in Chapter 17, “Class of Service to Cache Mask Association: Common Across Allocation Features”.

17.19.4.4 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.COS 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 17-34 shows the bit field layout for this register. Bits[63:32] contain the COS 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 COS value in IA32_PQR_ASSOC.COS greater than MAX_COS_CDP =(CPUID.(EAX=10H, ECX=1):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.COS is never written then the CAT capability defaults to using COS 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 17.19.7, "Introduction to Memory Bandwidth Allocation" for important COS programming considerations including maximum values when using CAT and CDP.

17.19.5 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.(EAX=10H, ECX=1):ECX.CDP[bit 2] (see Figure 17-32). Most of the CPUID.(EAX=10H, ECX=1) sub-leaf data that applies to CAT also apply to CDP. However, CPUID.(EAX=10H, ECX=1):EDX.COS_MAX_CAT specifies the maximum COS applicable to CAT-only operation. For CDP operations, COS_MAX_CDP is equal to (CPUID.(EAX=10H, ECX=1):EDX.COS_MAX_CAT >>1).

If CPUID.(EAX=10H, ECX=1):ECX.CDP[bit 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 17-36. 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.COS is COS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

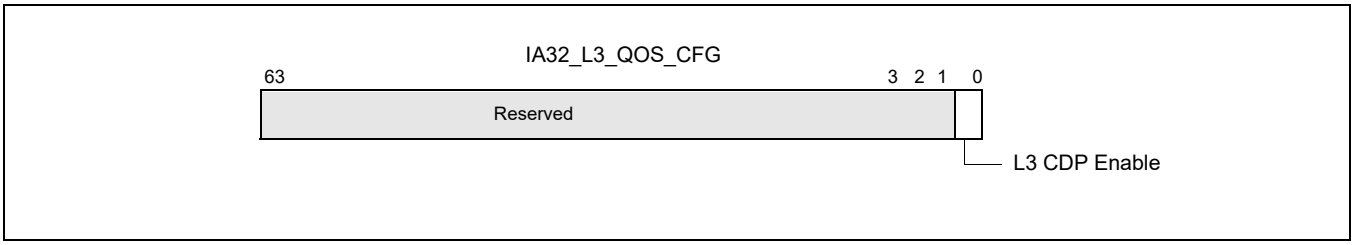


Figure 17-36. Layout of IA32_L3_QOS_CFG

IA32_L3_QOS_CFG default values are all 0s at RESET, the mask MSRs are all 1s. Hence, all logical processors are initialized in COS0 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 17.19.7 for software considerations while enabling or disabling L3 CDP.

17.19.5.1 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 COS. The re-mapping is shown in Table 17-19.

Table 17-19. Re-indexing of COS Numbers and Mapping to CAT/CDP Mask MSRs

Mask MSR	CAT-only Operation	CDP Operation
IA32_L3_QOS_Mask_0	COS0	COS0.Data
IA32_L3_QOS_Mask_1	COS1	COS0.Code
IA32_L3_QOS_Mask_2	COS2	COS1.Data
IA32_L3_QOS_Mask_3	COS3	COS1.Code
IA32_L3_QOS_Mask_4	COS4	COS2.Data
IA32_L3_QOS_Mask_5	COS5	COS2.Code
….	….	….
IA32_L3_QOS_Mask_'2n'	COS'2n'	COS'n'.Data
IA32_L3_QOS_Mask_'2n+1'	COS'2n+1'	COS'n'.Code

One can derive the MSR address for the data mask or code mask for a given COS number 'n' by:

- data_mask_address (n) = base + (n <<1), where base is the address of IA32_L3_QOS_MASK_0.
- code_mask_address (n) = base + (n <<1) +1.

When CDP is enabled, each COS 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 17-29).

Mask MSR field definitions remain the same. Capacity masks must be formed of contiguous set bits, 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).

17.19.6 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.(EAX=10H, ECX=2):ECX.CDP[bit 2] (see Figure 17-33). Most of the CPUID.(EAX=10H, ECX=2) sub-leaf data that applies to CAT also apply to CDP. However, CPUID.(EAX=10H, ECX=2):EDX.COS_MAX_CAT specifies the maximum COS applicable to CAT-only operation. For CDP operations, COS_MAX_CDP is equal to (CPUID.(EAX=10H, ECX=2):EDX.COS_MAX_CAT >>1).

If CPUID.(EAX=10H, ECX=2):ECX.CDP[bit 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 17-37. 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.COS is COS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

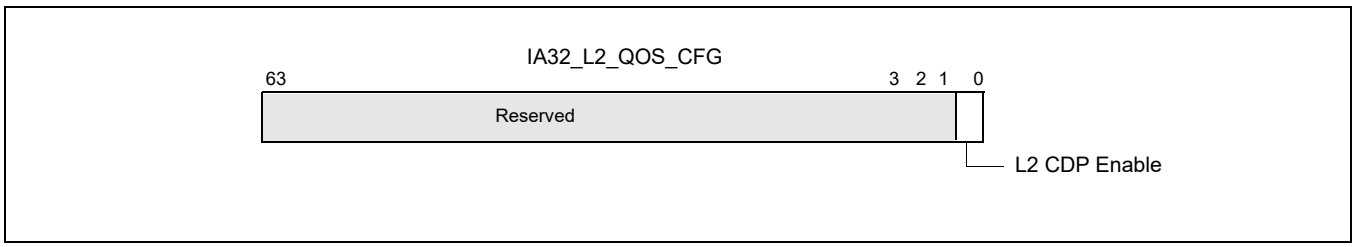


Figure 17-37. Layout of IA32_L2_QOS_CFG

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 COS0 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.

17.19.6.1 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 COS. This remapping is the same as the remapping shown in Table 17-19 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 COS 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 17-29).

Mask MSR field definitions for L2 CDP remain the same as for L2 CAT. Capacity masks must be formed of contiguous set bits, 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).

17.19.6.2 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 COS number which are valid in CDP operation is used, otherwise undefined behavior may result. For instance in a case with 16 CAT COS, since COS are reduced by half when CDP is enabled, software should ensure that only COS 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 COS 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 COS[0] before enabling or disabling CDP.

17.19.6.3 Cache Allocation Technology Dynamic Configuration

All Resource Director Technology (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 COS (the max COS number is specified in CPUID.(EAX=10H, ECX=ResID):EDX[15:0]), or
- Writing a COS greater than the supported maximum (specified as the maximum value of CPUID.(EAX=10H, ECX=ResID):EDX[15:0] for all valid ResID values) is written to the IA32_PQR_ASSOC.CLOS field.

When CDP is enabled, specifying a COS value in IA32_PQR_ASSOC.COS outside of the lower half of the COS 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 COS on the core will be returned.

When reading an IA32_*resourceType*_MASK_*n* register the current capacity bit mask for COS 'n' will be returned.

As noted previously, software should minimize migrations of COS across logical processors (across threads or cores), as a reduction in the accuracy of the Cache Allocation feature may result if COS 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 COS migration across processor logical threads and processor cores.

17.19.6.4 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.

17.19.6.5 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 COS with a dedicated partition in the cache. Such an SMM handler can switch to the dedicated COS immediately upon entering SMM, and switching back to the previously running COS upon exit.

17.19.6.6 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 COS concept applies to both CAT and CDP (for instance, COS[5] means the same thing whether CAT or CDP is in use, and the COS has associated resource usage constraint attributes including cache capacity masks). The mapping of COS 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 COS 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 COS number, one for code, one for data.
  - Masks for code/data are interleaved in the MSR address space (see Table 17-19).

17.19.7 Introduction to Memory Bandwidth Allocation

The Memory Bandwidth Allocation (MBA) feature provides indirect and approximate control over memory bandwidth available per-core, and was introduced on the Intel Xeon Processor Scalable Family. This feature provides a method to control applications which may be over-utilizing bandwidth relative to their priority in environments such as the data-center.

The MBA feature uses existing constructs from the Resource Director Technology (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 leaf 10H) are shared.

The high-level implementation of Memory Bandwidth Allocation is shown in Figure 17-38.

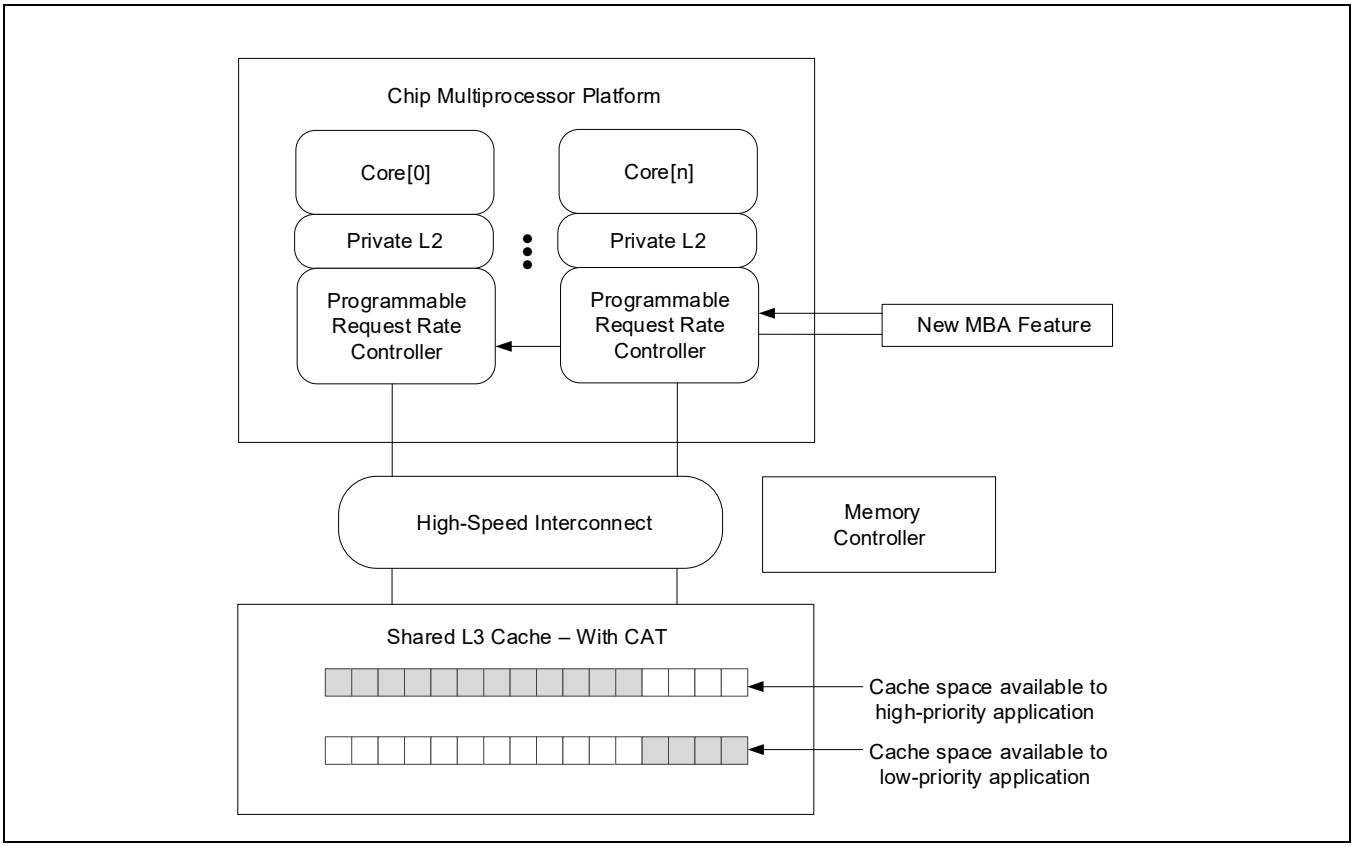


Figure 17-38. A High-Level Overview of the MBA Feature

As shown in Figure 17-38, the MBA feature introduces a programmable request rate controller between the cores and the high-speed interconnect, enabling indirect control over memory bandwidth for cores over-utilizing bandwidth relative to their priority. For instance, high-priority cores may be run un-throttled, but lower priority cores generating an excessive amount of traffic may be throttled to enable more bandwidth availability for the high-priority cores.

Since MBA uses a programmable rate controller between the cores and the interconnect, higher-level shared caches and memory controller, bandwidth to these caches may also be reduced, so care should be taken to throttle only bandwidth-intense applications which do not use the off-core caches effectively.

The throttling values exposed by MBA are approximate, and are calibrated to specific traffic patterns. As work-load characteristics vary, the throttling values provided may affect each workload differently. In cases where precise control is needed, the Memory Bandwidth Monitoring (MBM) feature can be used as input to a software controller which makes decisions about the MBA throttling level to apply.

Enumeration and configuration details are discussed below followed by usage model considerations.

17.19.7.1 Memory Bandwidth Allocation Enumeration

Similar to other 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 RDT features which enable control over shared platform resources is enumerated by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.(EAX=07H, ECX=0):EBX.PQE[bit 15] reports 1, the processor supports software control over shared processor resources. Software may then use CPUID leaf 10H to enumerate additional details on the specific controls provided.

Through CPUID leaf 10H software may determine whether MBA is supported on the platform. Specifically, as shown in Figure 17-31, 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 17-38 which provides the following details.

- CPUID.(EAX=10H, ECX=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.(EAX=10H, ECX=ResID=3):EBX is reserved.
- CPUID.(EAX=10H, ECX=ResID=3):ECX[2] reports whether the response of the delay values is linear (see text).
- CPUID.(EAX=10H, ECX=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 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 RDT resource control features present on the platform.

Discussion on the interpretation and usage of the MBA delay values is provided in Section 17.19.7.2 on MBA configuration.

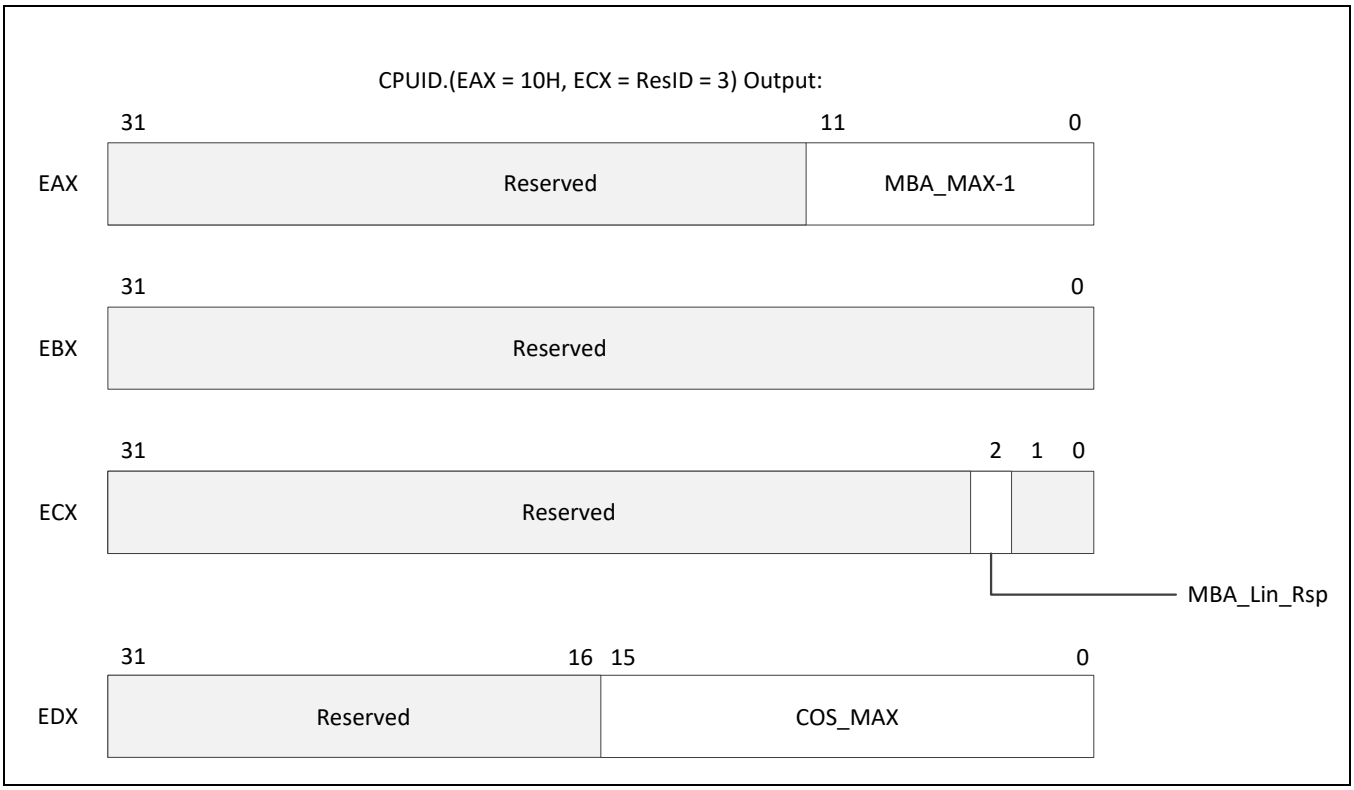


Figure 17-39. CPUID.(EAX=10H, ECX=3H) MBA Feature Details Identification

17.19.7.2 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 RDT features as described in Section 17.19.7.1 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 17.19.7.3 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 17-20.

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 17.19.7.1. 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 17-20 the base address of the MSRs is at D50H, and the range corresponds to the maximum supported CLOS from CPUID.(EAX=10H, ECX=ResID=1):EDX[15:0] as described in Section 17.19.7.1. For instance, if 16 CLOS are supported then the valid MSR range will extend from D50H through D5F inclusive.

Table 17-20. 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_'COS_MAX'	D50H + COS_MAX from CPUID.10H.3

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.(EAX=10H, ECX=ResID=3):ECX[bit 2] then values from zero through the MBA_MAX field from CPUID.(EAX=10H, ECX=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.(EAX=10H, ECX=ResID=3):ECX[bit 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.

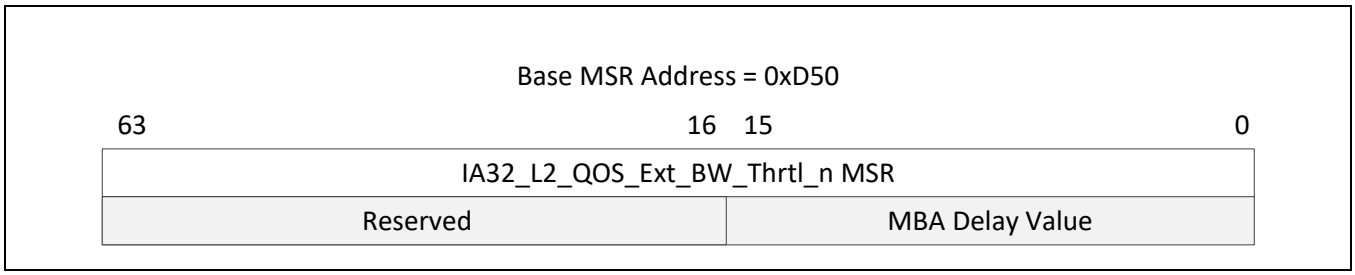


Figure 17-40. IA32_L2_QoS_Ext_BW_Thrtl_n MSR Definition

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.

17.19.7.3 Memory Bandwidth Allocation Usage Considerations

As the memory bandwidth control that MBA provides is indirect and approximate, using the feature with a closed-loop controller to also monitor memory bandwidth and how effectively the applications use the cache (via the Cache Monitoring Technology feature) may provide additional value. This approach also allows administrators to provide a band-width target or set-point which a controller could use to guide MBA throttling values applied, and this allows bandwidth control independent of the execution characteristics of the application.

As control is provided per processor core (the max of the delay values of the per-thread CLOS applied to the core) care should be taking in scheduling threads so as to not inadvertently place a high-priority thread (with zero intended MBA throttling) next to a low-priority thread (with MBA throttling intended), which would lead to inadver-tent throttling of the high-priority thread.

18. New Chapter 18, Volume 3B

A new Chapter 18 has been added to the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

This chapter describes the Last Branch Records (architectural) in detail.

CHAPTER 18
LAST BRANCH RECORDS

NOTE

This chapter defines a last-branch recording (LBR) facility that is architectural and part of the Intel 64 architecture. This facility is an enhancement of but distinct from earlier LBR facilities that were not architectural. Those earlier facilities are documented in Chapter 17.

Last Branch Records (LBRs) enable recording of software path history by logging taken branches and other control flow transfers within processor registers. Each LBR record or entry is comprised of three MSRs:

- IA32_LBR_x_FROM_IP — Holds the source IP of the operation.
- IA32_LBR_x_TO_IP — Holds the destination IP of the operation.
- IA32_LBR_x_INFO — Holds metadata for the operation, including mispredict, TSX, and elapsed cycle time information.

The number of LBR records available varies across processor generations, and is specified in CPUID.

LBR records are stored in age order. The most recent LBR entry is stored in IA32_LBR_0_*, the next youngest in IA32_LBR_1_*, and so on. When an operation to be recorded completes (retires) with LBRs enabled (IA32_LBR_CTL.LBREn=1), older LBR entries are shifted in the LBR array by one entry, then a record of the new operation is written into entry 0. See Section 18.1.1 for the list of recorded operations.

The number of LBR entries available for recording operations is dictated by the value in IA32_LBR_DEPTH.DEPTH. By default, the DEPTH value matches the maximum number of LBRs supported by the processor, but software may opt to use fewer in order to achieve reduced context switch latency.

In addition to the LBRs, there is a single Last Event Record (LER). It records the last taken branch preceding the last exception, hardware interrupt, or software interrupt. Like LBRs, the LER is comprised of three MSRs (IA32_LER_FROM_IP, IA32_LER_TO_IP, IA32_LER_INFO), and is subject to the same dependencies on enabling and filtering.

Which operations are recorded in LBRs depends upon a series of factors:

- Branch Type Filtering — Software must opt in to the types of branches to be logged; see Section 18.1.2.3.
- Current Privilege Level (CPL) — LBRs can be filtered based on CPL; see Section 18.1.2.5.
- LBR Freeze — LBR and LER recording can be suspended by setting IA32_PERF_GLOBAL_STATUS.LBR_FRZ to 1. See Section 17.4.7 for details on LBR_FRZ.

On some implementations, recording LBRs may require constraining the number of operations that can complete in a cycle. As a result, on these implementations, enabling LBRs may have some performance overhead.

18.1 BEHAVIOR

18.1.1 Logged Operations

LBRs can log most control flow transfer operations.

The source IP recorded for a branch instruction is the IP of that instruction. For events that take place between instructions, the source IP recorded is the IP of the next sequential instruction.

The destination IP recorded is always the target of the branch or event, the next instruction that will execute.

The full list of operations and the respective IPs recorded is shown in Table 18-1.

Table 18-1. LBR IP Values for Various Operations

Operation	FROM_IP	TO_IP
Taken Branch[1], Exception, INT3, INTn, INTO, TSX Abort	Current IP	Target IP
Interrupt	Next IP	Target IP
INIT (BSP)	Next IP	Reset Vector
INIT (AP) + SIPI	Next IP	SIPI Vector
EENTER/ERESUME + EEXIT/AEX	Current IP	Target or Trampoline IP
RSM[2]	Target IP	Target IP
#DB, #SMI, VM exit, VM entry	None	None

NOTES:

1. Direct CALLs with displacement zero, for which the target is typically the next sequential IP, are not treated as taken branches by LBRs.
2. RSM is only recorded in LBRs when IA32_DEBUGCTL.FREEZE_WHILE_SMM is set to 0.

18.1.2 Configuration

18.1.2.1 Enabling and Disabling

LBRs are enabled by setting IA32_LBR_CTL.LBREn to 1.
Some operations, such as entry to a secure mode like SMM or Intel SGX, can cause LBRs to be temporarily disabled. Other operations, such as debug exceptions or some SMX operations, disable LBRs and require software to re-enable them. Details on these interactions can be found in Section 18.1.4.

18.1.2.2 LBR Depth

The number of LBRs used by the processor can be constrained by modifying the IA32_LBR_DEPTH.DEPTH value. DEPTH defaults to the maximum number of LBRs supported by the processor. Allowed DEPTH values can be found in CPUID.(EAX=01CH, ECX=0):EAX[7:0].

Reducing the LBR depth can result in improved performance, by reducing the number of LBRs that need to be read and/or context switched.

On a software write to IA32_LBR_DEPTH, all LBR entries are reset to 0. LERs are not impacted.

A RDMSR or WRMSR to any IA32_LBR_x_* MSRs, such that x ≥ DEPTH, will generate a #GP exception. Note that the XSAVES and XRSTORS instructions access only the LBRs associated with entries 0 to DEPTH-1.

By clearing the LBR entries on writes to IA32_LBR_DEPTH, and forbidding any software writes to LBRs ≥ DEPTH, it is thereby guaranteed that any LBR entries equal to or above DEPTH will have value 0.

18.1.2.3 Branch Type Enabling and Filtering

Software must opt in to the types of branches that are desired to be recorded. These elections are made in IA32_LBR_CTL; see Section 18.2. Branch type options are listed in Table 18-2; only those enabled will be recorded.

Table 18-2. Branch Type Filtering Details

Branch Type	Operations Recorded
COND	Jcc, J*CXZ, and LOOP*
NEAR_IND_JMP	JMP r/m*
NEAR_REL_JMP	JMP rel*
NEAR_IND_CALL	CALL r/m*
NEAR_REL_CALL	CALL rel* (excluding CALLs to the next sequential IP)
NEAR_RET	RET (0C3H)
OTHER_BRANCH	JMP/CALL ptr*, JMP/CALL m*, RET (0C8H), SYS*, interrupts, exceptions (other than debug exceptions), IRET, INT3, INTn, INTO, TSX Abort, EENTER, ERESUME, EEXIT, AEX, INIT, SIPI, RSM

These encodings match those in IA32_LBR_x_INFO.BR_TYPE.

Control flow transfers that are not recorded include #DB, VM exit, VM entry, and #SMI.

18.1.2.4 Call-Stack Mode

The LBR array is, by default, treated as a ring buffer that captures control flow transitions. However, the finite depth of the LBR array can be limiting 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. These calls to leaf functions, and their returns, are likely to displace the main execution context from the LBRs.

When call-stack mode is enabled, the LBR array can capture unfiltered call data normally, but as return instructions are executed the last captured branch (call) record is flushed from the LBRs in a last-in first-out (LIFO) manner. Thus, branch information pertaining to completed leaf functions will not be retained, while preserving the call stack information of the main line execution path.

Call-stack mode is enabled by setting IA32_LBR_CTL.CALL_STACK to 1. When enabled, near RET instructions receive special treatment. Rather than adding a new record in LBR_0, a near RET will instead "pop" the CALL entry at LBR_0 by shifting entries LBR_1..LBR_[DEPTH-1] up to LBR_0..LBR_[DEPTH-2], and clearing LBR_[DEPTH-1] to 0. Thus, LBR processing software can consume only valid call-stack entries by reading until finding an entry that is all zeros.

Call-stack mode should be used with branch type enabling configured to capture only CALLs (NEAR_REL_CALL and NEAR_IND_CALL) and RETs (NEAR_RET). When configured in this manner, the LBR array emulates a call stack, where CALLs are "pushed" and RETs "pop" them off the stack. If other branch types (JCC, NEAR_*_JMP, or OTHER_BRANCH) are enabled for recording with call-stack mode, LBR behavior may be undefined.

It is recommended that call-stack mode be used along with CPL filtering, by setting at most one of the OS and USR bits in the IA32_LBR_CTL MSR. Call-stack mode does not emulate the stack switch that can occur on CPL transitions, and hence monitoring all CPLs may result in a corrupted LBR call stack.

Call-Stack Mode and LBR Freeze

When IA32_DEBUGCTL.FREEZE_LBRS_ON_PMI=1, IA32_PERF_GLOBAL_STATUS.LBR_FRZ will be set to 1 when a PMI is pended. That will cause LBRs and LERs to cease recording branches until LBR_FRZ is cleared. Because there may be some "skid", or instructions retiring, in between the PMI being pended and the PMI being taken, it is possible that some branches may be missing from the LBRs. In the case of call-stack mode, if a CALL or RET is missed, that can lead to confusing results where CALL entries fail to get "popped" off the stack, and RETs "pop" the wrong CALLs.

An alternative is to utilize CPL filtering to limit LBR recording to less privileged modes only (CPL>3) instead of using the FREEZE_LBRS_ON_PMI=1 feature. This will record branches in the "skid", but avoid recording any branches in the privilege level 0 handler.

18.1.2.5 CPL Filtering

Software must opt in to which CPL(s) will have branches recorded. If IA32_LBR_CTL.OS=1, then branches in CPL=0 can be recorded. If IA32_LBR_CTL.USR=1, then branches in CPL>0 can be recorded. For operations which change the CPL, the operation is recorded in LBRs only if the CPL at the end of the operation is enabled for LBR recording. In cases where the CPL transitions from a value that is filtered out to a value that is enabled for LBR recording, the FROM_IP address for the recorded CPL transition branch or event will be 0FFFFFFFFFFFFFFFH.

18.1.3 Record Data

18.1.3.1 IP Fields

The source and destination IP values in IA32_LBR_x_[FROM|TO]_IP and IA32_LER_x_[FROM|TO]_IP may hold effective IPs or linear IPs (LIPs), depending on the processor generation. The effective IP is the offset from the CS base address, while LIP includes the CS base address. Which IP type is used is indicated in CPUID.(EAX=01CH, ECX=0):EAX[bit 31].

The value read from this field will always be canonical. Note that this includes the case where a canonical violation (#GP) results from executing sequential code that runs precisely to the end of the lower canonical address space (where IP[63:MAXLINADDR-1] is 0, but IP[MAXLINADDR-2:0] is all ones). In this case, the FROM_IP will hold the lowest canonical address in the upper canonical space, such that IP[63:MAXLINADDR-1] is all ones, and IP[MAXLINADDR-2:0] is 0.

In some cases, due to CPL filtering, the FROM_IP of the recorded operation may be filtered out. In this case 0FFFFFFFFFFFFFFFH will be recorded. See Section 18.1.2.5 for details.

Writes of these fields will be forced canonical, such that the processor ignores the value written to the upper bits (IP[63:MAXLINADDR-1]).

18.1.3.2 Branch Types

The IA32_LBR_x_INFO.BR_TYPE and IA32_LER_INFO.BR_TYPE fields encode the branch types as shown in Table 18-3.

Table 18-3. IA32_LBR_x_INFO and IA32_LER_INFO Branch Type Encodings

Encoding	Branch Type
0000B	COND
0001B	NEAR_IND_JMP
0010B	NEAR_REL_JMP
0011B	NEAR_IND_CALL
0100B	NEAR_REL_CALL
0101B	NEAR_RET
011xB	Reserved
1xxxB	OTHER_BRANCH

For a list of branch operations that fall into the categories above, see Table 18-2. In future generations, BR_TYPE bits 2:0 may be used to distinguish between differing types of OTHER_BRANCH.

18.1.3.3 Cycle Time

Each time an operation is recorded in an LBR, the value of the LBR cycle timer is recorded in IA32_LBR_x_INFO.CYC_CNT. The LBR cycle timer is a saturating counter that counts at the processor clock rate. Each time an operation is recorded in an LBR, the counter is reset but continues counting.

There is an LBR cycle counter valid bit, IA32_LBR_x_INFO.CYC_CNT_VALID. When set, the CYC_CNT field holds a valid value, the number of elapsed cycles since the last operation recorded in an LBR (up to 0FFFFH).

Some implementations may opt to reduce the granularity of the CYC_CNT field for larger values. The implication of this is that the least significant bits may be forced to 1 in cases where the count has reached some minimum threshold. It is guaranteed that this reduced granularity will never result in an inaccuracy of more than 10%.

18.1.3.4 Mispredict Information

IA32_LBR_x_INFO.MISPRED provides an indication of whether the recorded branch was predicted incorrectly by the processor. The bit is set if either the taken/not-taken direction of a conditional branch was mispredicted, or if the target of an indirect branch was mispredicted.

18.1.3.5 Intel® TSX Information

IA32_LBR_x_INFO.IN_TSX indicates whether the operation recorded retired during a TSX transaction. IA32_LBR_x_INFO.TSX_ABORT indicates that the operation is a TSX Abort.

18.1.4 Interaction with Other Processor Features

18.1.4.1 SMM

IA32_LBR_CTL.LBREn is saved and cleared on #SMI, and restored on RSM. As a result of disabling LBRs, the #SMI is not recorded. RSM is recorded only if IA32_DEBUGCTL.FREEZE_WHILE_SMM is set to 0, and the FROM_IP will be set to the same value as the TO_IP.

18.1.4.2 SMM Transfer Monitor (STM)

LBREn is not cleared on #SMI when it causes SMM VM exit. Instead, the STM should use the VMCS controls described in Section 18.1.4.3 to disable LBRs while in SMM, and to restore them on VM entries that exit SMM.

On VMCALL to configure STM, IA32_LBR_CTL is cleared.

18.1.4.3 VMX

By default, LBR operation persists across VMX transitions. However, VMCS fields have been added to enable constraining LBR usage to within non-root operation only. See details in Table 18-4.

Table 18-4. LBR VMCS Fields

Name	Type	Bit Position	Behavior
Guest IA32_LBR_CTL	Guest State Field	NA	The guest value of IA32_LBR_CTL is written to this field on all VM exits.
Load Guest IA32_LBR_CTL	Entry Control	21	When set, VM entry will write the value from the "Guest IA32_LBR_CTL" guest state field to IA32_LBR_CTL.
Clear IA32_LBR_CTL	Exit Control	26	When set, VM exit will clear IA32_LBR_CTL after the value has been saved to the "Guest IA32_LBR_CTL" guest state field.

To enable "guest-only" LBR use, a VMM should set both the "Load Guest IA32_LBR_CTL" entry control and the "Clear IA32_LBR_CTL" exit control. For "system-wide" LBR use, where LBRs remain enabled across host and guest(s), a VMM should keep both new VMCS controls clear.

VM entry checks that, if the "Load Guest IA32_LBR_CTL" entry control is 1, bits reserved in the IA32_LBR_CTL MSR must be 0 in the field for that register.

For additional information relating to VMX transitions, see Chapter 24, Chapter 26, and Chapter 27 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.

18.1.4.4 Intel® SGX

On entry to an enclave, via EENTER or ERESUME, logging of LBR entries is suspended. On enclave exit, via EEXIT or AEX, logging resumes. The cycle counter will continue to run during enclave execution.

An exception to the above is made for opt-in debug enclaves. For such enclaves, LBR logging is not impacted.

18.1.4.5 Debug Exceptions

When a branch happens because of a #DB exception, IA32_LBR_CTL.LBREn is cleared. As a result, the operation is not recorded.

18.1.4.6 SMX

On GETSEC leaves SENTER or ENTERACCS, IA32_LBR_CTL is cleared. As a result, the operation is not recorded.

18.1.4.7 MWAIT

On an MWAIT that requests a C-state deeper than C1, IA32_LBR_x_* MSRs may be cleared to 0. IA32_LBR_CTL, IA32_LBR_DEPTH, and IA32_LER_* MSRs will be preserved.

For an MWAIT that enters a C-state equal to or less deep than C1, and all C-states that enter as a result of Hardware Duty Cycling (HDC), all LBR MSRs are preserved.

18.1.4.8 Processor Event-Based Sampling (PEBS)

PEBS records can be configured to include LBRs, by setting PEBS_DATA_CFG.LBREn[3]=1. The number of LBRs to include in the record is also configurable, via PEBS_DATA_CFG.NUM_LBRS[28:24]. For details on PEBS, see Section 19.9 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*.

If NUM_LBRS is set to a value greater than LBR_DEPTH, then only LBR_DEPTH entries will be written into the PEBS record. Further, the Record Size field will be decreased to match the actual size of the record to be written, and the Record Format field will replace the value of NUM_LBRS with the value of LBR_DEPTH. These adjustments ensure that software is able to properly interpret the PEBS record.

18.2 MSRS

The MSRs that represent the LBR entries (IA32_LBR_x_[TO|FROM|INFO]) and the LER entry (IA32_LER_[TO|FROM|INFO]) do not fault on writes. Any address field written will force sign-extension based on the maximum linear address width supported by the processor, and any non-zero value written to undefined bits may be ignored such that subsequent reads return 0.

On a warm reset, all LBR MSRs, including IA32_LBR_DEPTH, have their values preserved. However, IA32_LBR_CTL.LBREn is cleared to 0, disabling LBRs. If a warm reset is triggered while the processor is in the C6 idle state, also known as warm init, all LBR MSRs will be reset to their initial values.

See Table 2-2 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4* for details on LBR MSRs.

18.3 FAST LBR READ ACCESS

XSAVES provides a faster means than RDMSR for software to read all LBRs. When using XSAVES for reading LBRs rather than for context switch, software should take care to ensure that XSAVES does not write LBR state to an area of memory that has been or will be used by XRSTORS. This could corrupt INIT tracking.

18.4 OTHER IMPACTS

18.4.1 Branch Trace Store on Intel Atom® Processors

Branch Trace Store (BTS) on Intel Atom processors that support the architectural form of the LBR feature has dependencies on the LBR configuration. BTS will store out the LBR_0 (TOS) record each time a taken branch or event retires. If any filtering of LBRs is employed, or if LBRs are disabled, some duplicate entries may be stored by BTS. Like LBRs and LERs, BTS is suspended when IA32_PERF_GLOBAL_STATUS.LBR_FRZ is set to 1.

BTS will change to cease issuing branch records for direct near CALLs with displacement zero to align with LBR behavior.

18.4.2 IA32_DEBUGCTL

On processors that do not support model-specific LBRs, IA32_DEBUGCTL[bit 0] has no meaning. It can be written to 0 or 1, but reads will always return 0.

18.4.3 IA32_PERF_CAPABILITIES

On processors that do not support model-specific LBRs, IA32_PERF_CAPABILITIES.LBR_FMT will have the value 03FH.

19. Updates to Chapter 19, Volume 3B

Change bars and green text show changes to Chapter 19 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

Changes to this chapter: Added new Section 19.3.10, "12th Generation Intel® Core™ Processor Performance Monitoring Facility". Updated Section 19.5.3.1.2, "Reduced Skid PEBS", Section 19.9.2.2.2, "Memory Access Info", Section 19.9.4, "Reduced Skid PEBS", and Section 19.9.5, "EPT-Friendly PEBS". Added new Section 19.9.6, "Precise Distribution (PDIST)", Section 19.9.7, "Load Latency Facility", and Section 19.9.8, "Store Latency Facility". Numerous updates, typo corrections, and additions/updates for clarification added throughout chapter.

CHAPTER 19
PERFORMANCE MONITORING

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 hosted by the Intel Open Source Technology Center. These files can be downloaded here: https://download.01.org/perfmon/.

19.1 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 19.6.3, "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/.

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 19.2.

See also:

- Section 19.2, "Architectural Performance Monitoring".
- Section 19.3, "Performance Monitoring (Intel® Core™ Processors and Intel® Xeon® Processors)".
  - Section 19.3.1, "Performance Monitoring for Processors Based on Nehalem Microarchitecture".
  - Section 19.3.2, "Performance Monitoring for Processors Based on Westmere Microarchitecture".
  - Section 19.3.3, "Intel® Xeon® Processor E7 Family Performance Monitoring Facility".
  - Section 19.3.4, "Performance Monitoring for Processors Based on Sandy Bridge Microarchitecture".
  - Section 19.3.5, "3rd Generation Intel® Core™ Processor Performance Monitoring Facility".

19.2 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.

19.2.1 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:

- Bit field layout of IA32_PERFEVTSELx is consistent across microarchitectures.
- 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[bits 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.

19.2.1.1 Architectural Performance Monitoring Version 1 Facilities

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 19-1 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 19-1, 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.

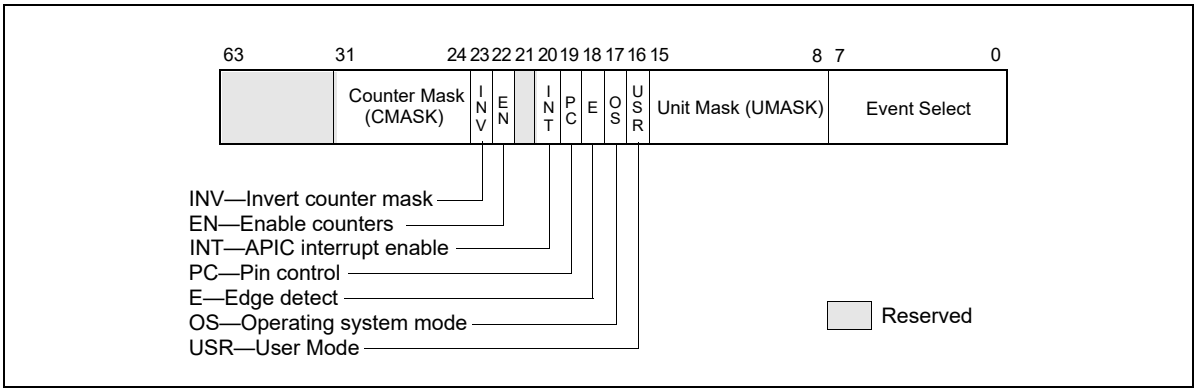


Figure 19-1. Layout of IA32_PERFEVTSELx MSRs

- **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 19-1; 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 PM*i* pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PM*i* 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.

19.2.1.2 Pre-defined Architectural Performance Events

Table 19-1 lists architecturally defined events.

Table 19-1. UMask and Event Select Encodings for Pre-Defined Architectural Performance Events

Bit Position CPUID.AH.EBX	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

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 19-1). 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 14, “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 precisians on functionality should not be based on the event values or dependent on the existence of performance monitoring events.

19.2.2 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 17.4.7 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 17.4.7 for details of the legacy Freeze LBRs on PMI control.

The facilities provided by architectural performance monitoring version 2 can be queried from CPUID leaf 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 19-2 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:

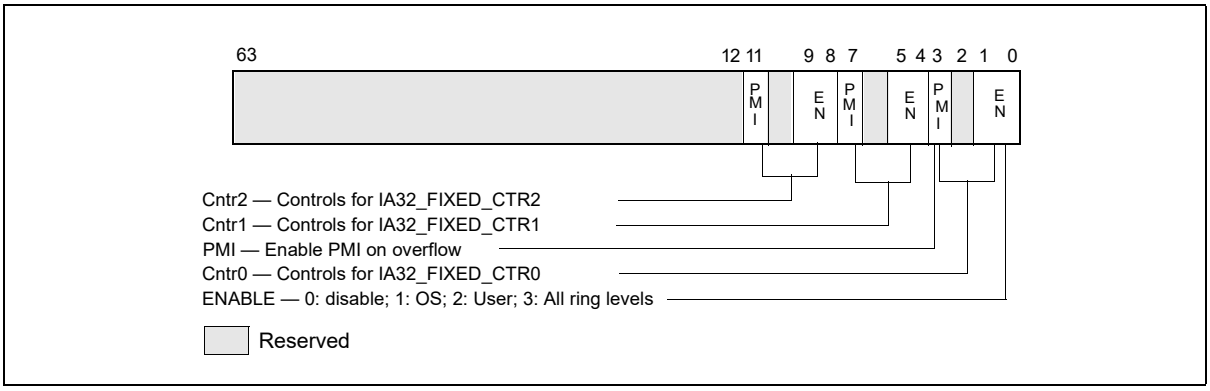


Figure 19-2. Layout of IA32_FIXED_CTR_CTRL MSR

- **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 19-3 shows the layout of IA32_PERF_GLOBAL_CTRL. Each enable bit in IA32_PERF_GLOBAL_CTRL is AND'ed 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 AND'ed results is true; counting is disabled when the result is false.

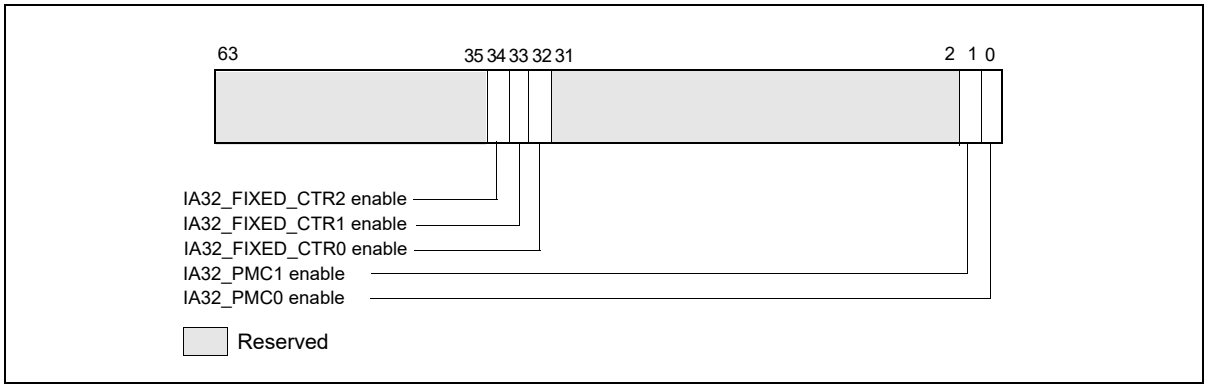


Figure 19-3. Layout of IA32_PERF_GLOBAL_CTRL MSR

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 19-2. 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.

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 19-4 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 (see Section 18.15.5), clears the counter overflow status., and sets the "OvfBuffer" bit in IA32_PERF_GLOBAL_STATUS.

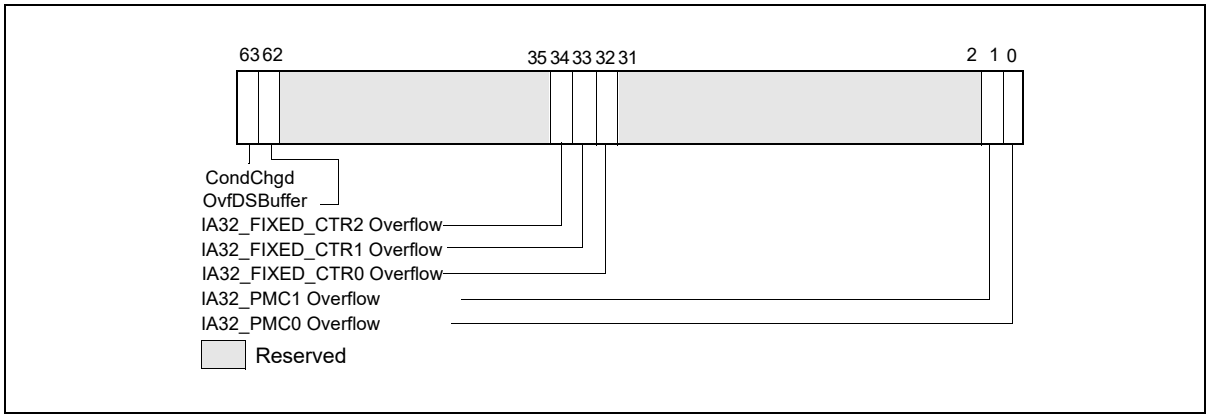


Figure 19-4. Layout of IA32_PERF_GLOBAL_STATUS MSR

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 19-5.

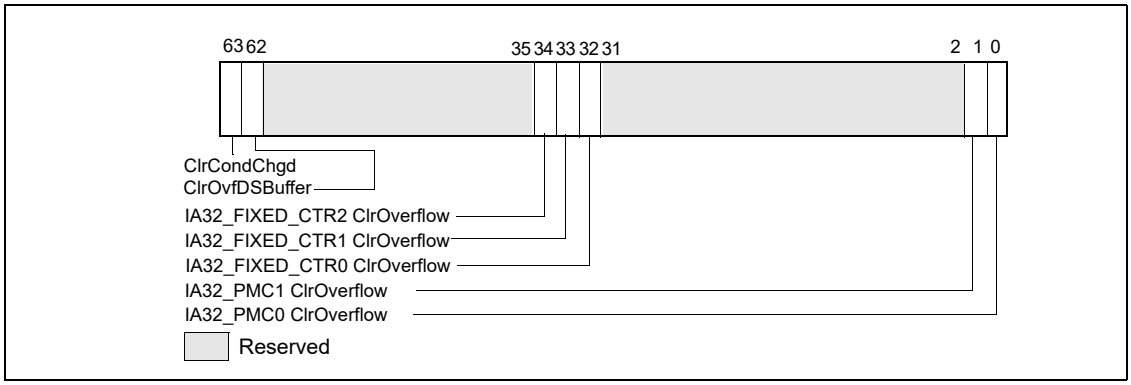


Figure 19-5. Layout of IA32_PERF_GLOBAL_OVF_CTRL MSR

19.2.3 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 leaf 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 19-6.

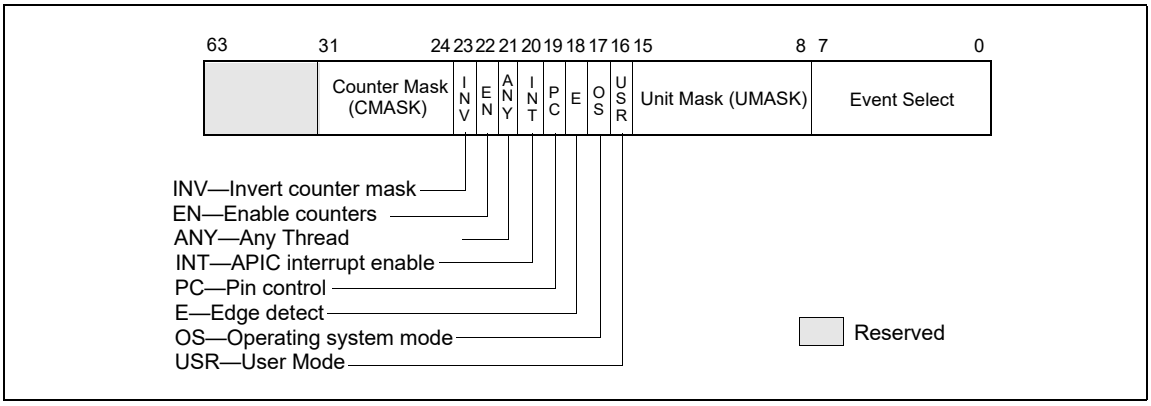


Figure 19-6. Layout of IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

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 allow thread-specificity configuration using an AnyThread bit. The layout of IA32_PERF_FIXED_CTR_CTRL MSR is shown.

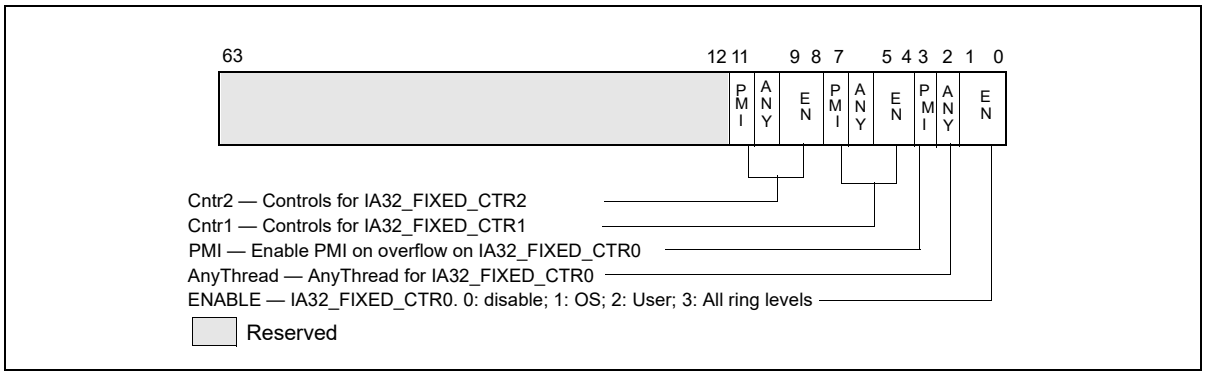


Figure 19-7. IA32_FIXED_CTR_CTRL MSR Supporting Architectural Performance Monitoring Version 3

Each control block for a fixed-function performance counter provides an **AnyThread** (bit position 2 + 4*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 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 19-8 and Figure 19-9 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 19-9) 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.

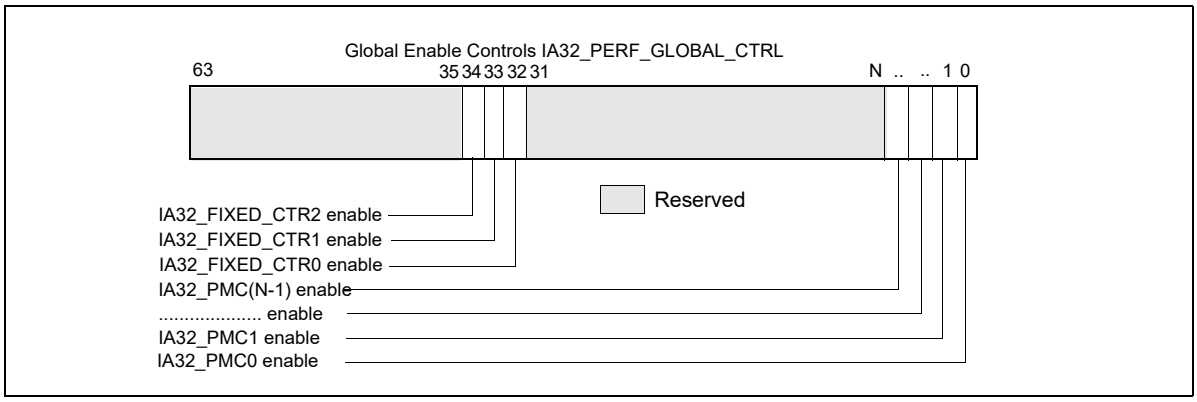


Figure 19-8. Layout of Global Performance Monitoring Control MSR

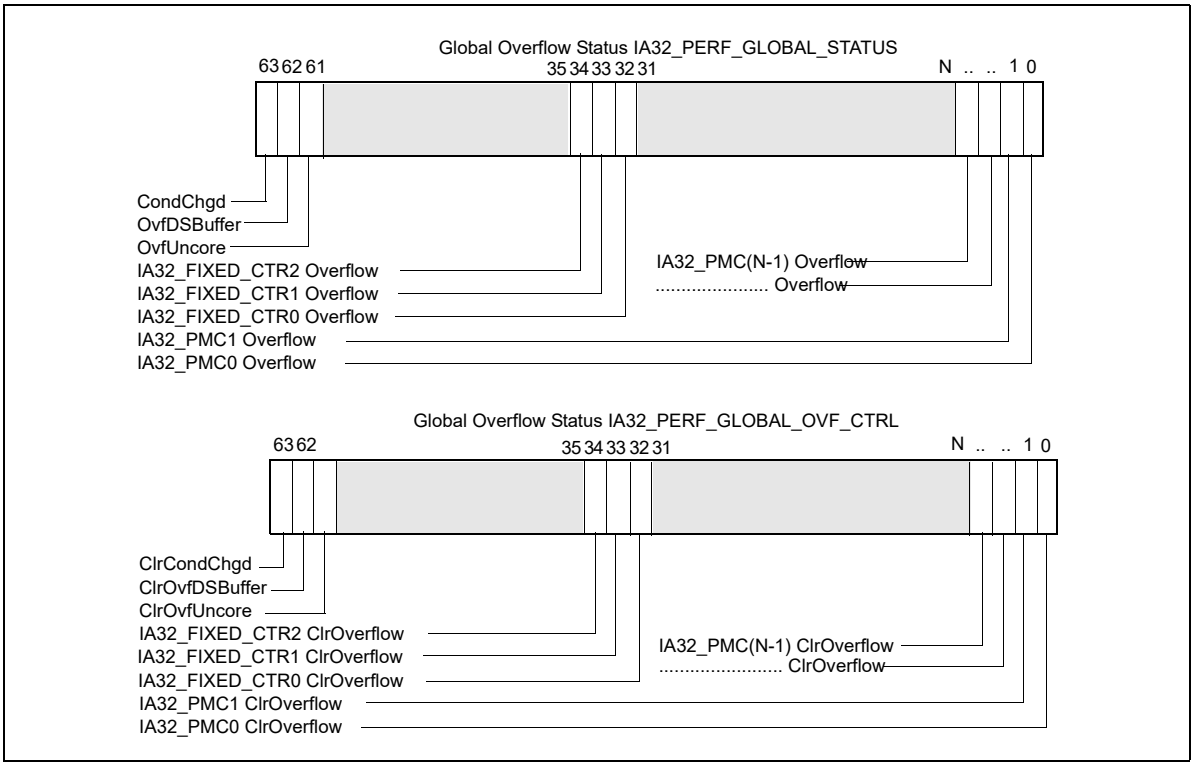


Figure 19-9. Global Performance Monitoring Overflow Status and Control MSRs

19.2.3.1 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 23, "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 24 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 24, CHAPTER 26, and CHAPTER 27).

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.

19.2.4 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 leaf 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 19.2.4.1.
- Streamlined Freeze/PMI Overhead management interfaces to use IA32_DEBUGCTL.Freeze_LBRs_On_PMI and IA32_DEBUGCTL.Freeze_PerfMon_On_PMI: see Section 19.2.4.1. 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 19.2.4.2.
- Performance monitoring resource in-use MSR to facilitate cooperative sharing protocol between perfmon-managing privilege agents.

19.2.4.1 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:
  - (IA32_DEBUGCTL.LBR & (!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:
    - (IA32_PERFEVTSELn.EN & IA32_PERF_GLOBAL_CTRL.PMCn & (!IA32_PERF_GLOBAL_STATUS.CTR_FRZ)) = 1 for programmable counter ‘n’, or
    - (IA32_PERF_FIXED_CRTL.ENi & IA32_PERF_GLOBAL_CTRL.FCi & (!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 17-3. 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 SGX (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).

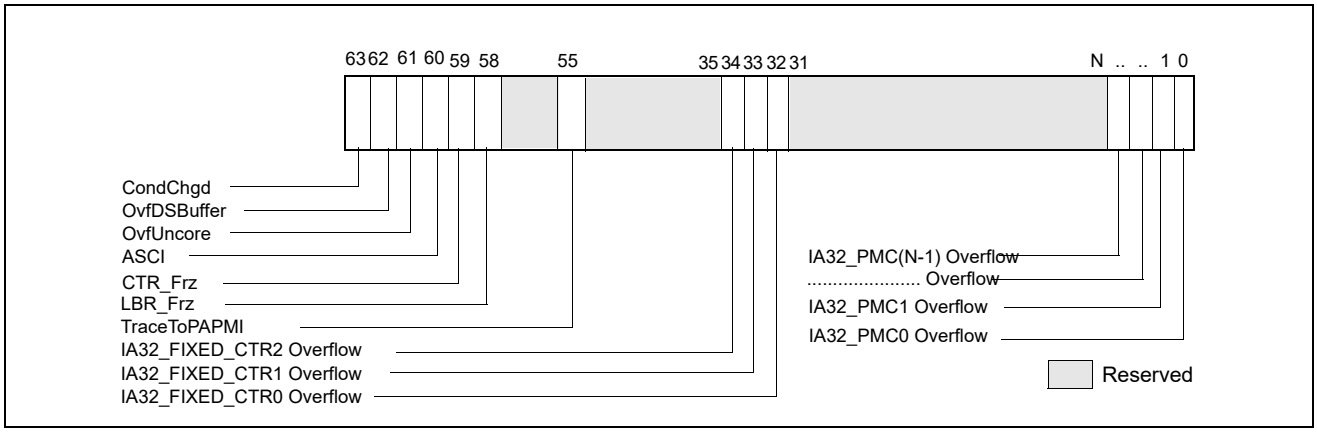


Figure 19-10. IA32_PERF_GLOBAL_STATUS MSR and Architectural Perfmon Version 4

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.

19.2.4.2 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 19-11). Further, IA32_PERF_GLOBAL_STATUS_RESET provides additional bit fields to clear the new indicators in IA32_PERF_GLOBAL_STATUS described in Section 19.2.4.1.

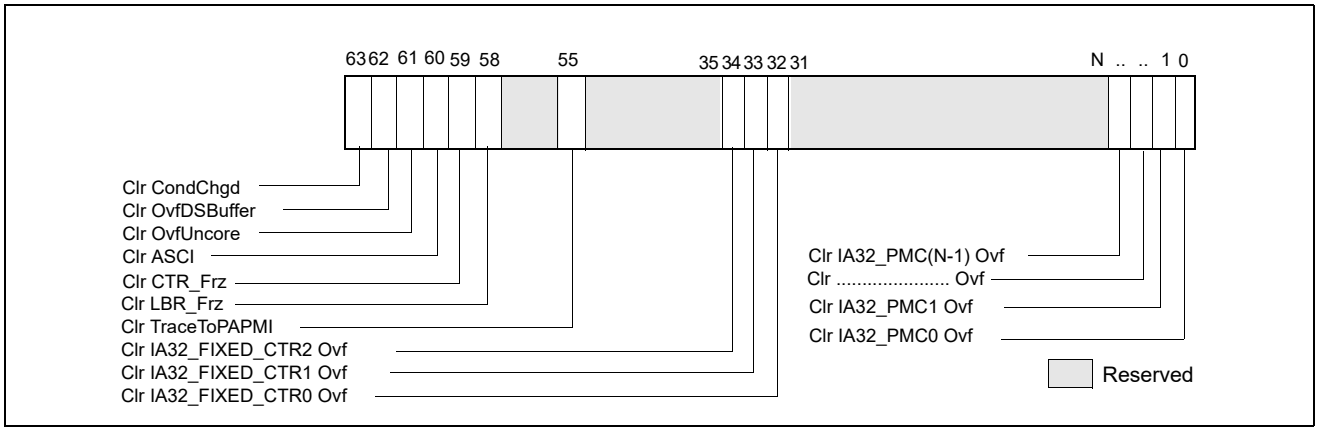


Figure 19-11. IA32_PERF_GLOBAL_STATUS_RESET MSR and Architectural Perfmon Version 4

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.

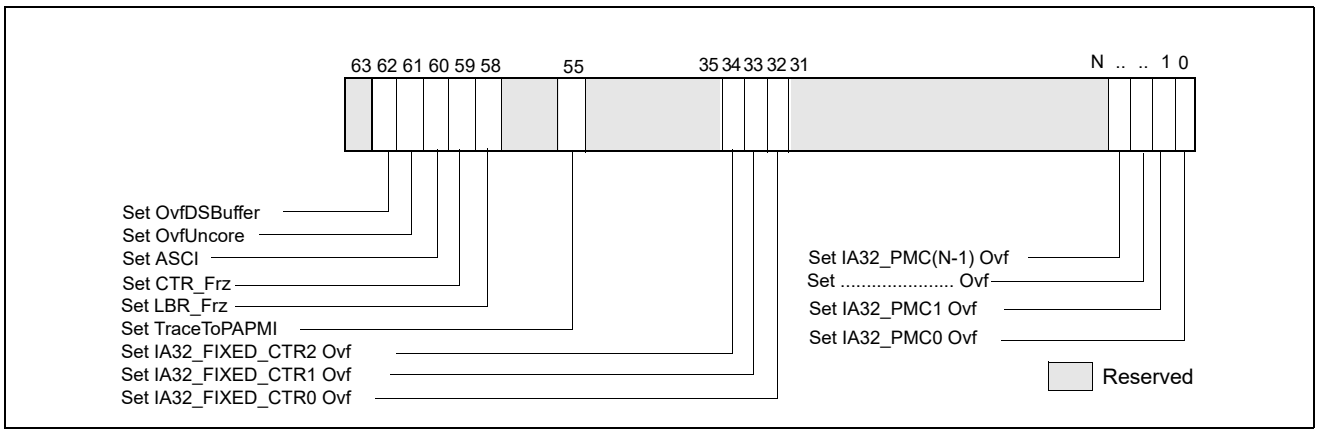


Figure 19-12. IA32_PERF_GLOBAL_STATUS_SET MSR and Architectural Perfmon Version 4

19.2.4.3 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 19-13.

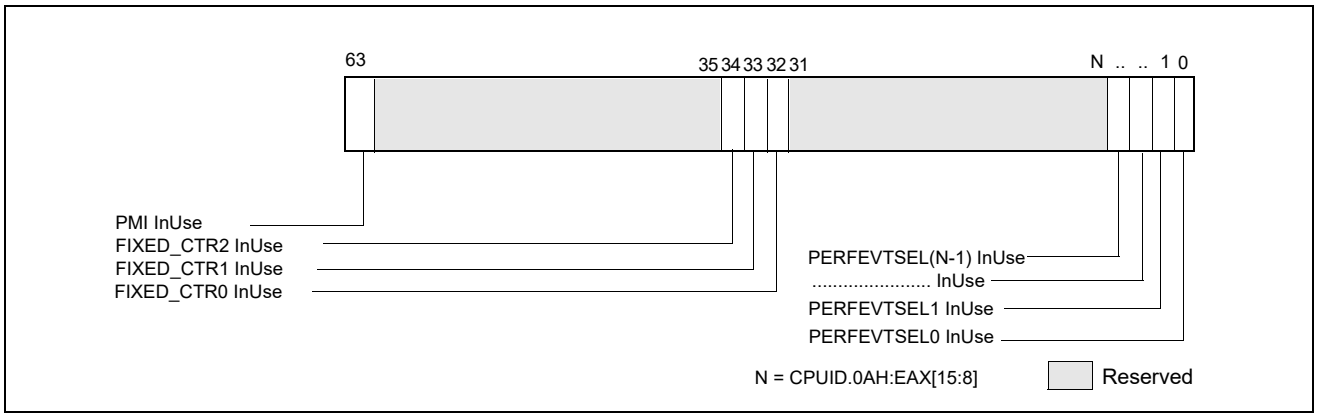


Figure 19-13. IA32_PERF_GLOBAL_INUSE MSR and Architectural Perfmon Version 4

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).
- 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.

1. Available at http://www.intel.com/sdm

19.2.5 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 leaf 0AH. Specifically, version 5 provides the following enhancements:

- Deprecation of AnyThread mode, see Section 19.2.5.1.
- Individual enumeration of Fixed counters in CPUID.0AH, see Section 19.2.5.2.
- Domain separation, see Section 19.2.5.3.

19.2.5.1 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 19.2.3.1.

19.2.5.2 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:

FxCtr[i]_is_supported := ECX[i] || (EDX[4:0] > i);

19.2.5.3 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.

19.2.6 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 19-65.

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
```

19.3 PERFORMANCE MONITORING (INTEL® CORE™ PROCESSORS AND INTEL® XEON® PROCESSORS)

19.3.1 Performance Monitoring for Processors Based on Nehalem Microarchitecture

Intel Core i7 processor family[1] supports architectural performance monitoring capability with version ID 3 (see Section 19.2.3) 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 19-6 and described in Section 19.2.1.1 and Section 19.2.3.

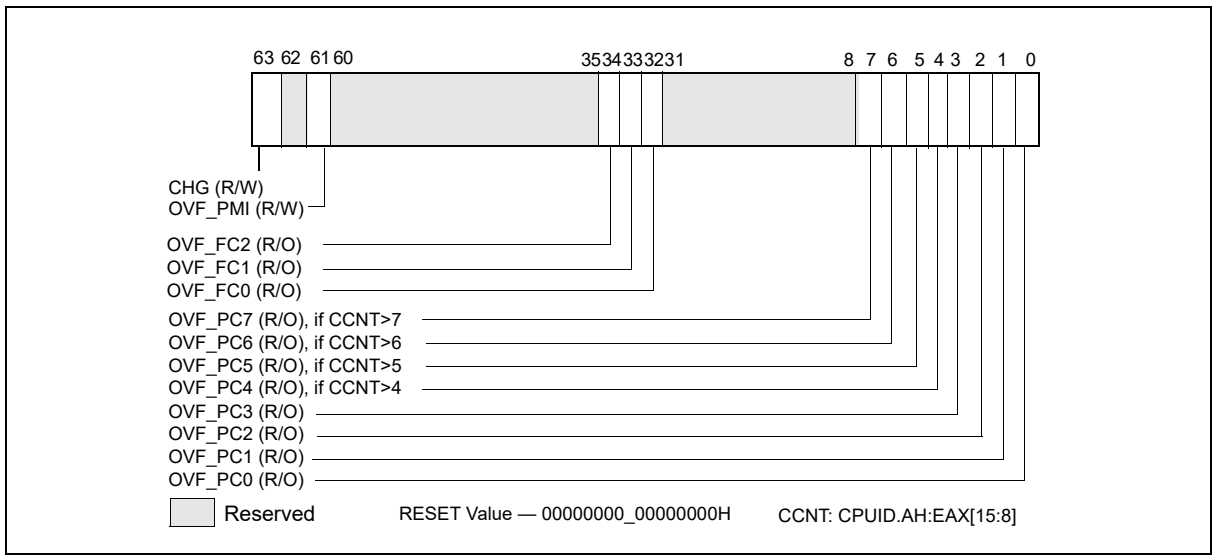


Figure 19-14. IA32_PERF_GLOBAL_STATUS MSR

19.3.1.1 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_PERFEVTSELx, 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

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.

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_PERFEVTSELx. Two off-core response MSRs are provided to use in conjunction with specific event codes that must be specified with IA32_PERFEVTSELx.

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 19.2.1.

19.3.1.1.1 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 19-15.

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.

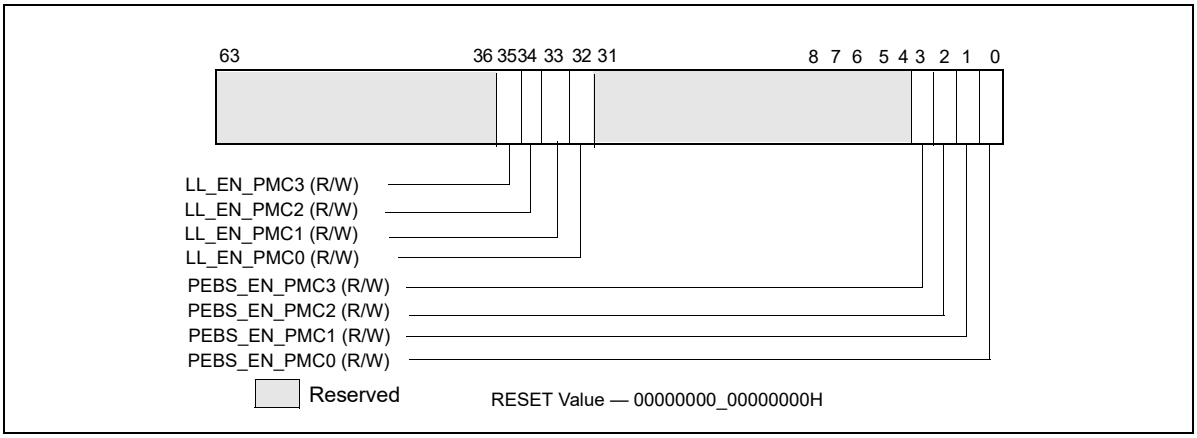


Figure 19-15. Layout of IA32_PEBS_ENABLE MSR

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 19-65).

The behavior of PEBS assists is reported by IA32_PERF_CAPABILITIES[6] (see Figure 19-65). 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 19-3, 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[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 19-3. 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 19-84. 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 19-16.

- **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.

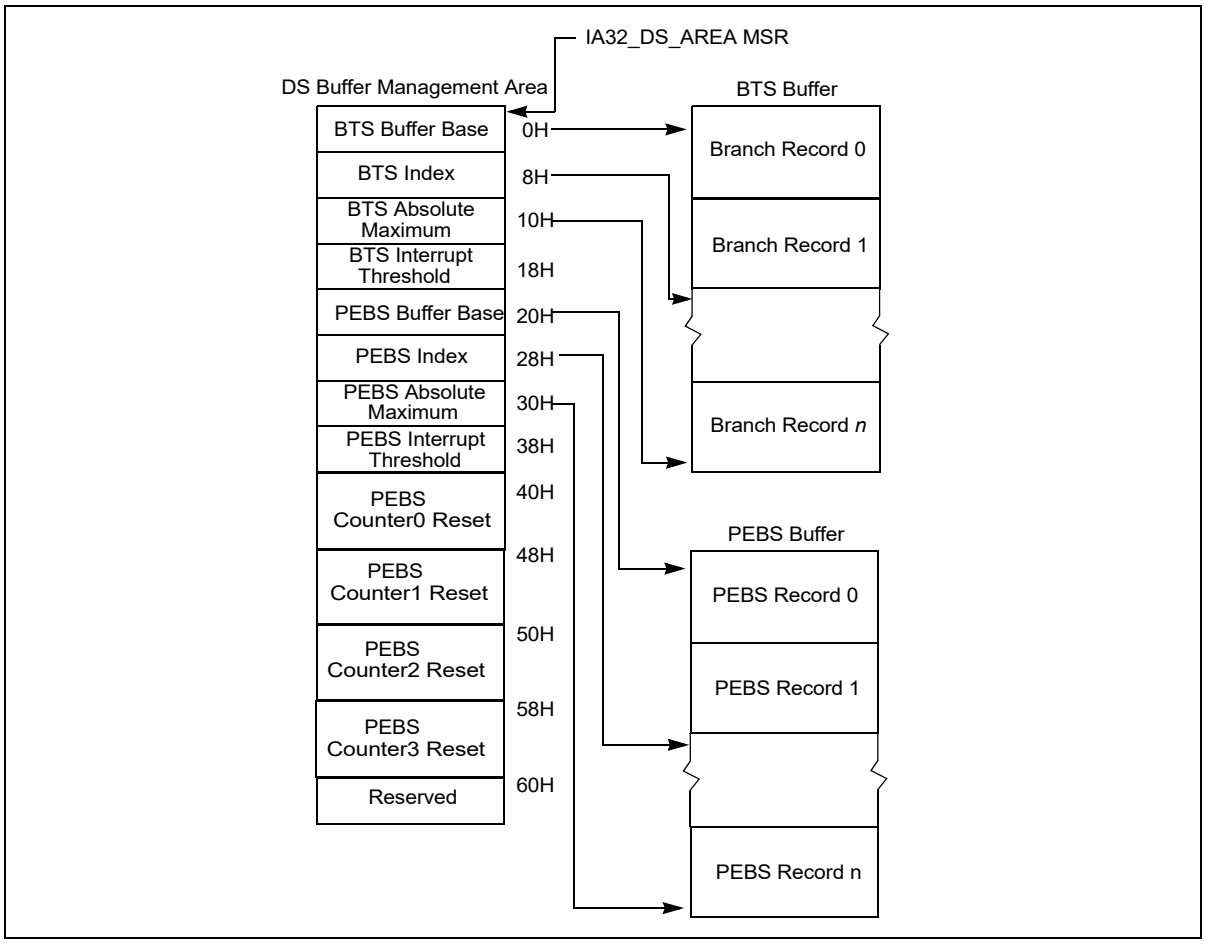


Figure 19-16. PEBS Programming Environment

- **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 record. 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 19.3.1.2). 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.

19.3.1.1.2 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 19-3. 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 19-3, 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 19-4. 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 19-4. 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 un-cacheable 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 19-17.

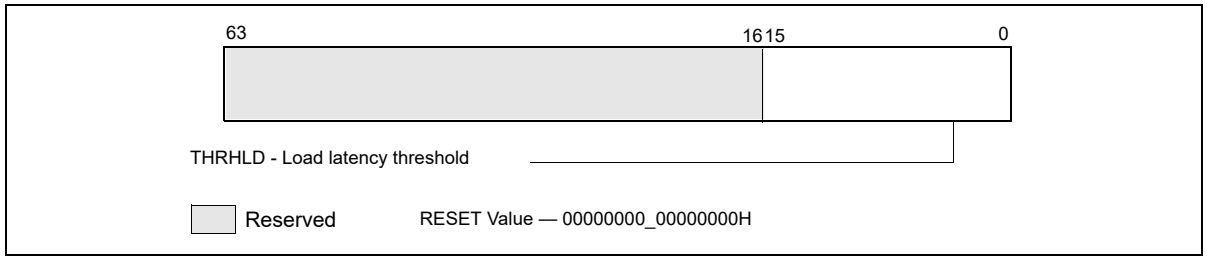


Figure 19-17. Layout of MSR_PEBS_LD_LAT MSR

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.

19.3.1.1.3 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 19-5 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 19-5. 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 19-18. Bits 7:0 specifies the request type of a transaction request to the uncore. Bits 15:8 specifies the response of the uncore subsystem.

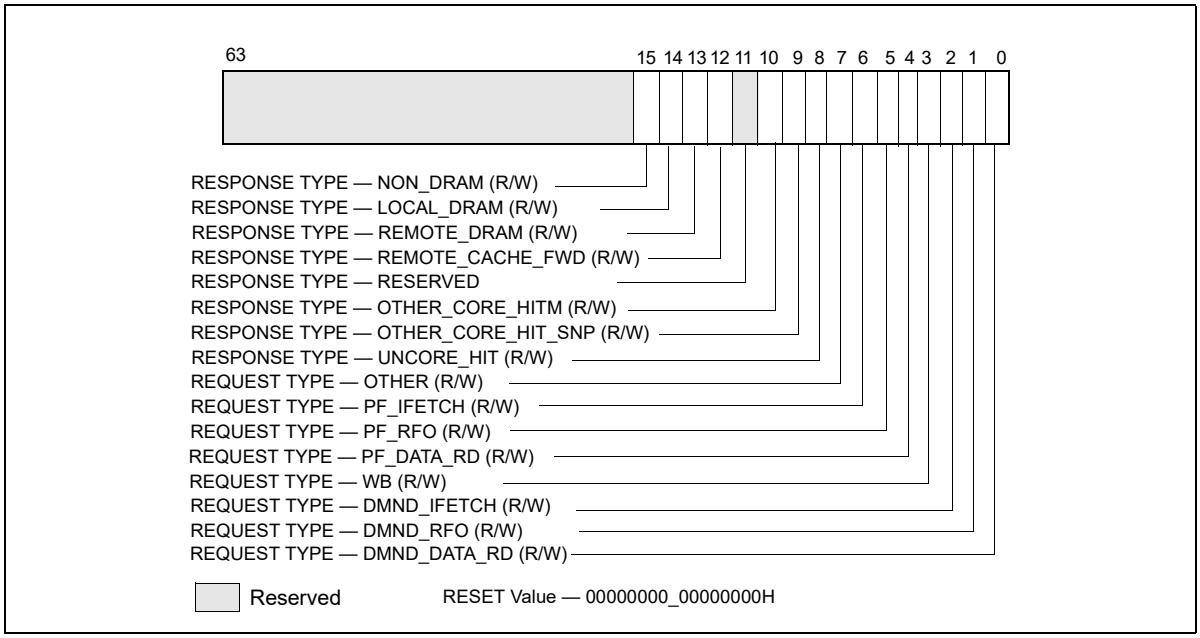


Figure 19-18. Layout of MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 to Configure Off-core Response Events

Table 19-6. 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.
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_HITM	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.

Table 19-6. MSR_OFFCORE_RSP_0 and MSR_OFFCORE_RSP_1 Bit Field Definition (Contd.)

Bit Name	Offset	Description
NON_DRAM	15	Non-DRAM requests that were serviced by IOH.

19.3.1.2 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 CPUID 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.

19.3.1.2.1 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 19-19 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_FixedCntr0.
- 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.

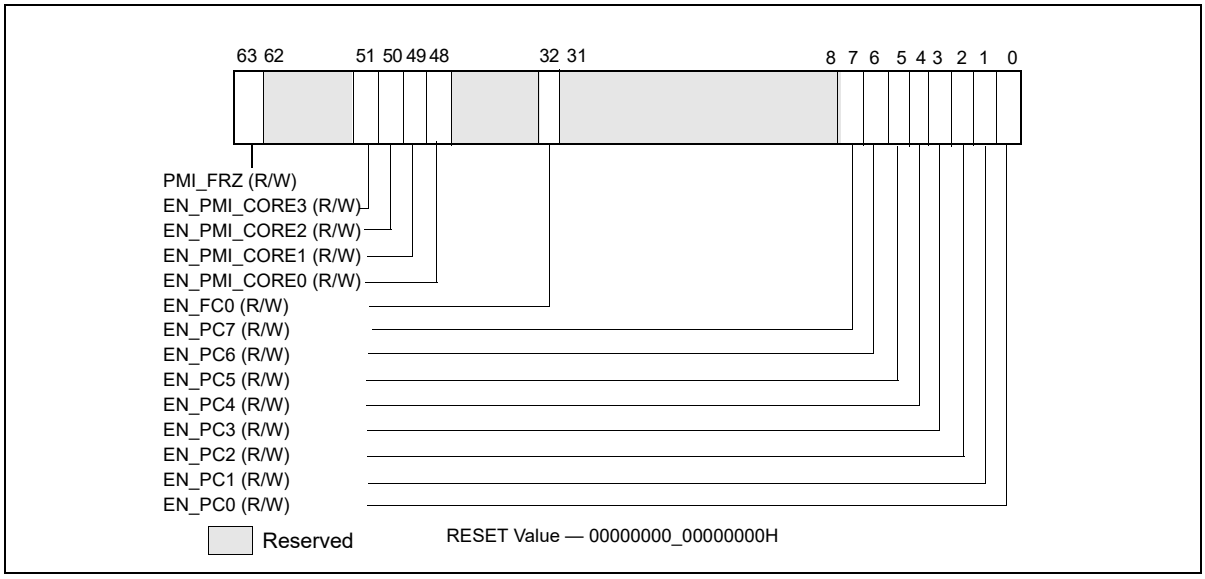


Figure 19-19. Layout of MSR_UNCORE_PERF_GLOBAL_CTRL MSR

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 19-20 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.

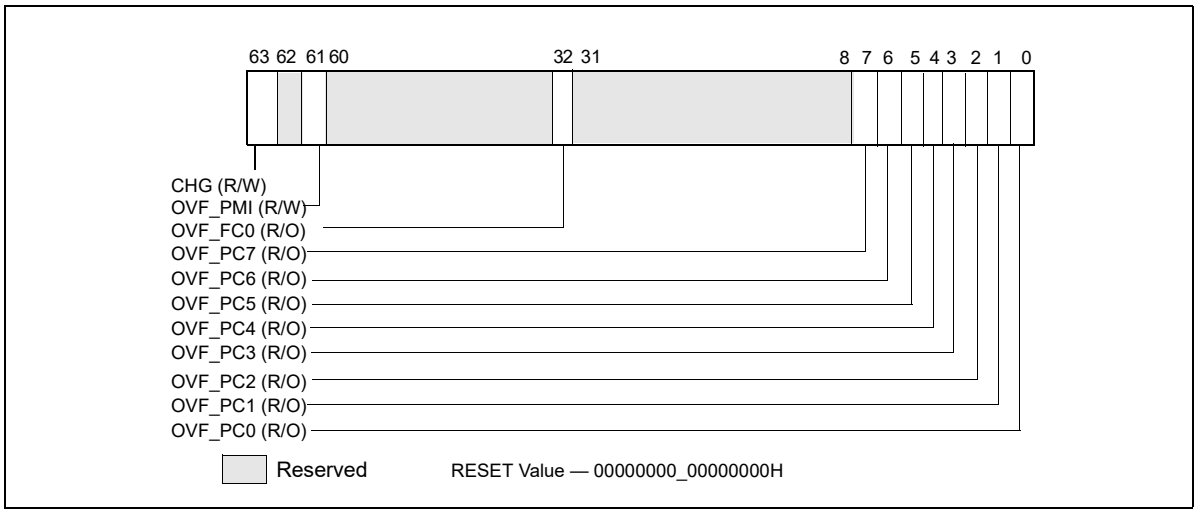


Figure 19-20. Layout of MSR_UNCORE_PERF_GLOBAL_STATUS MSR

Figure 19-21 shows the layout of MSR_UNCORE_PERF_GLOBAL_OVF_CTRL.

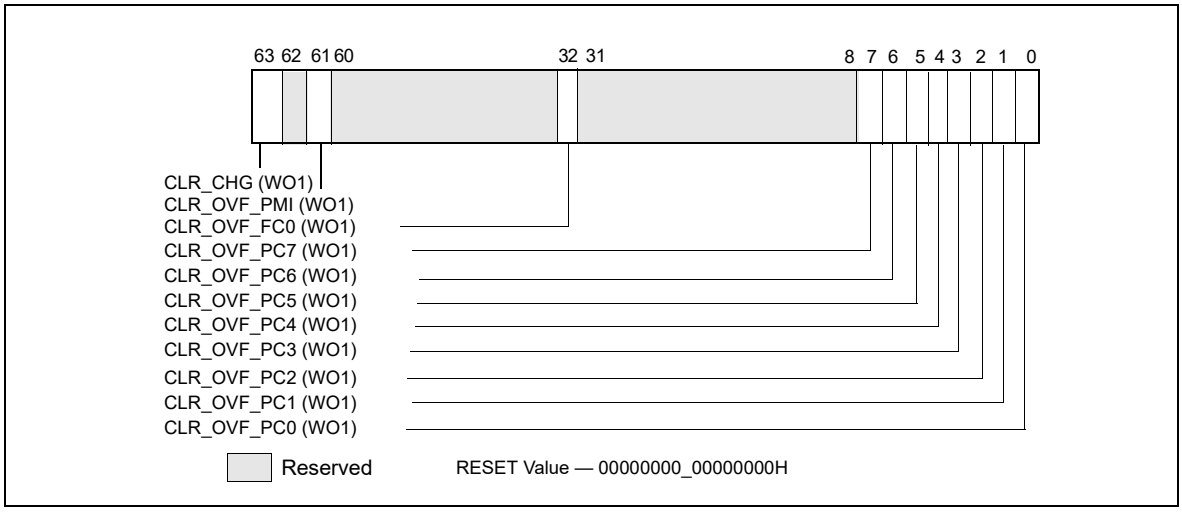


Figure 19-21. Layout of MSR_UNCORE_PERF_GLOBAL_OVF_CTRL MSR

- 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.

19.3.1.2.2 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 19-22 shows the layout of MSR_UNCORE_PERFEVTSELx.

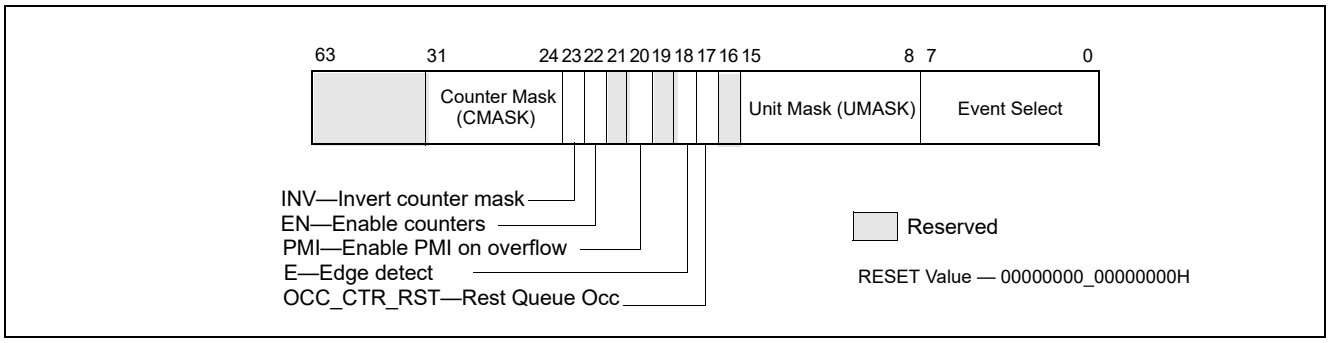


Figure 19-22. Layout of MSR_UNCORE_PERFEVTSELx MSRs

- 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 19-23 shows the layout of MSR_UNCORE_FIXED_CTR_CTRL.

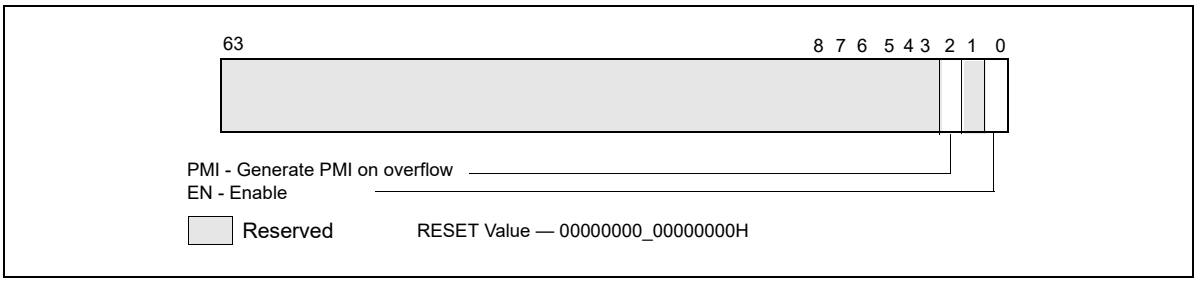


Figure 19-23. Layout of MSR_UNCORE_FIXED_CTR_CTRL MSR

- 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.

19.3.1.2.3 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 19-24.

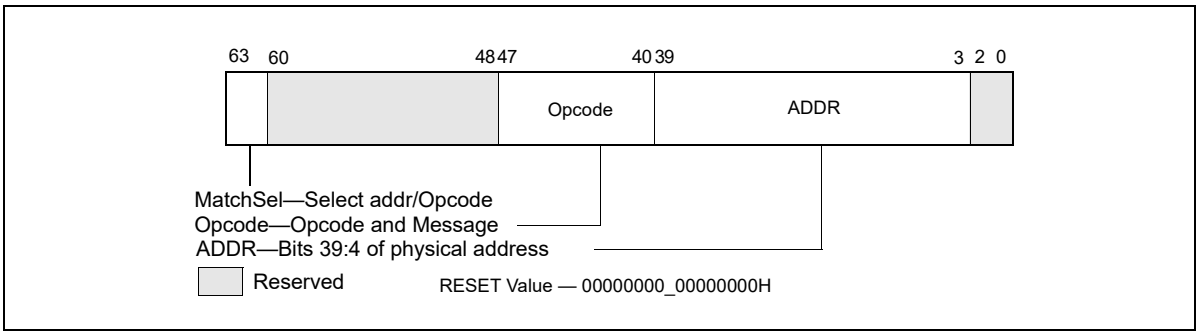


Figure 19-24. Layout of MSR_UNCORE_ADDR_OPCODE_MATCH MSR

- 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 19-7 lists the encodings supported in the opcode field.

Table 19-7. 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.

19.3.1.3 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 19-25.

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.

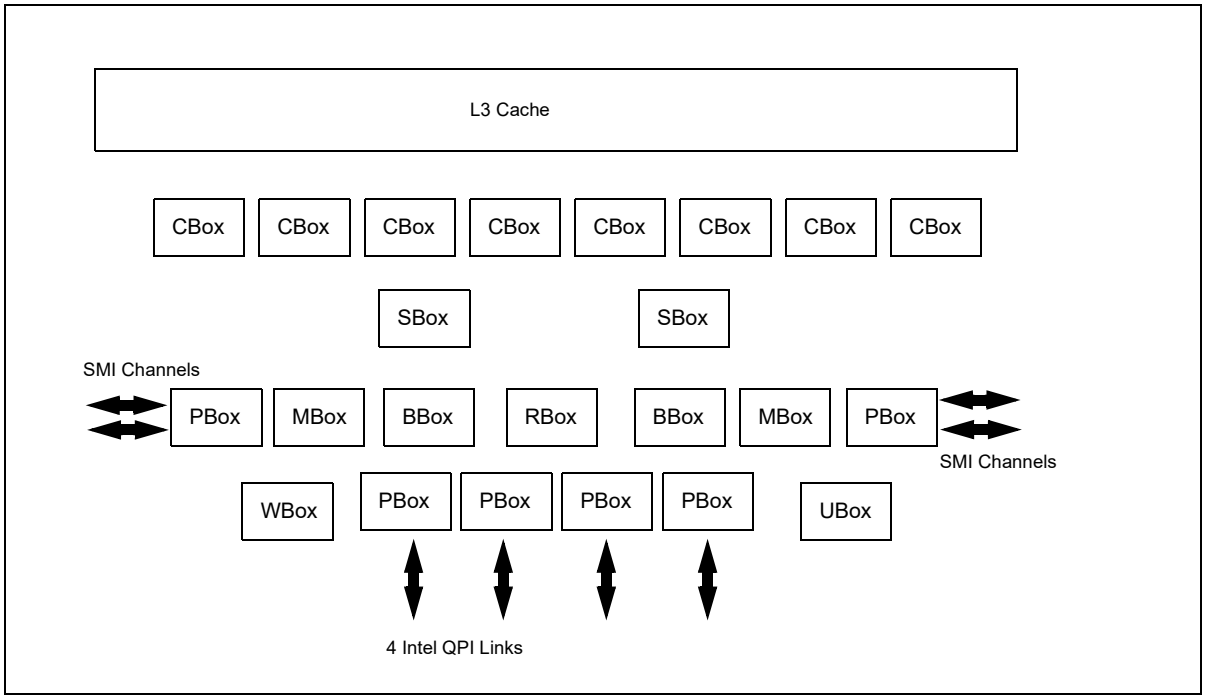


Figure 19-25. Distributed Units of the Uncore of Intel® Xeon® Processor 7500 Series

Table 19-8 summarizes the number MSRs for uncore PMU for each box.

Table 19-8. 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 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 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 the 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".

19.3.2 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 19.6.3 also apply to processors based on Westmere microarchitecture.

Table 19-5 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 19.3.1.1.2, 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 19-13.

19.3.3 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 series[1]. 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 19-25, with the additional capability that up to 10 C-Box units are supported.

Table 19-9 summarizes the number MSRs for uncore PMU for each box.

Table 19-9. 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".

19.3.4 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 19.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 19.2.3.

The core PMU's capability is similar to those described in Section 19.3.1.1 and Section 19.6.3, with some differences and enhancements relative to Westmere microarchitecture summarized in Table 19-10.

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 19-13.

Table 19-10. 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 19.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 19.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W:32	See Section 19.2.2.
# 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 19.2.1.
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 17.4.7.
Processor Event Based Sampling (PEBS) Events	See Table 19-12.	See Table 19-84.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 19.3.4.4.2; ▪ Data source encoding ▪ STLB miss encoding ▪ Lock transaction encoding	Data source encoding	
PEBS-Precise Store	Section 19.3.4.4.3	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.

19.3.4.1 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 19.2.1.1).

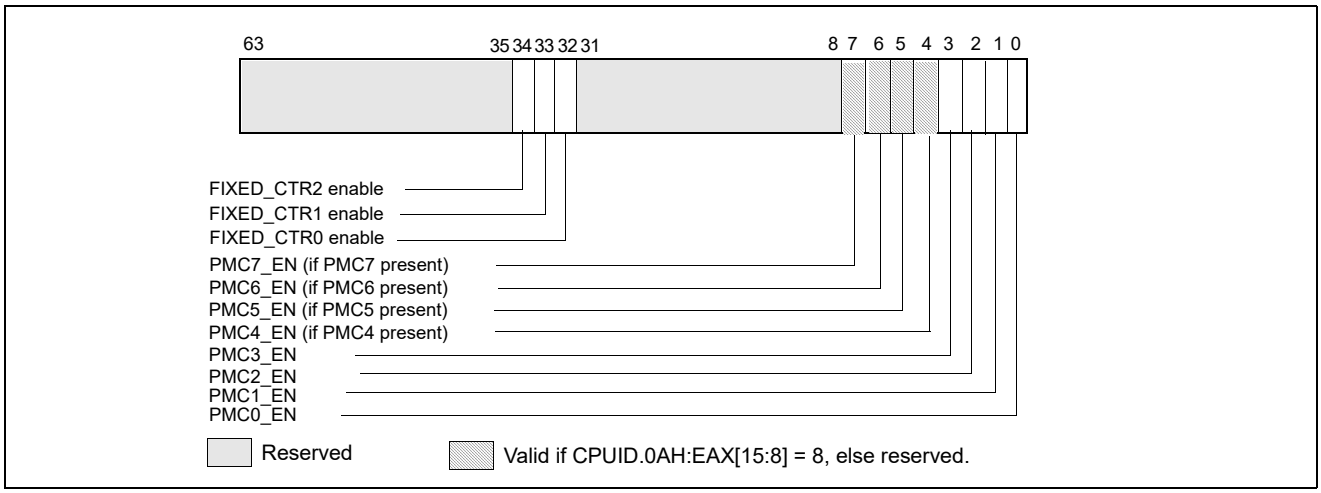


Figure 19-26. IA32_PERF_GLOBAL_CTRL MSR in Sandy Bridge Microarchitecture

Figure 19-44 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 AND'ed 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 AND'ed 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 19-27). A value of 1 in each bit of the PMCx_OVF field indicates an overflow condition has occurred in the associated counter.

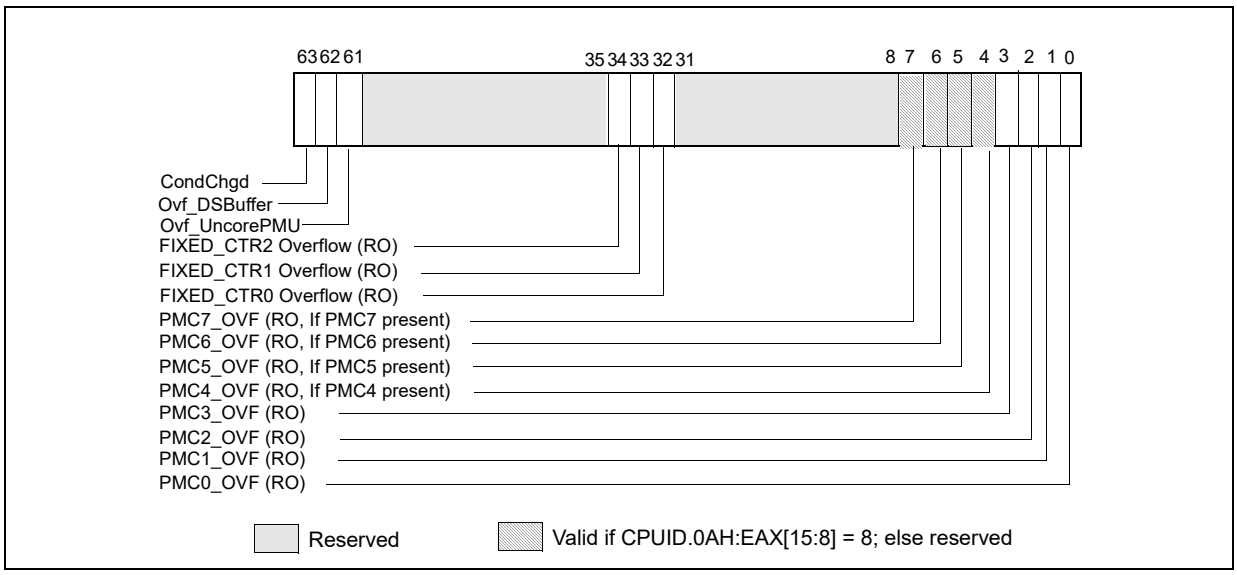


Figure 19-27. IA32_PERF_GLOBAL_STATUS MSR in Sandy Bridge Microarchitecture

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 17.4.9). 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 19-28). 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.

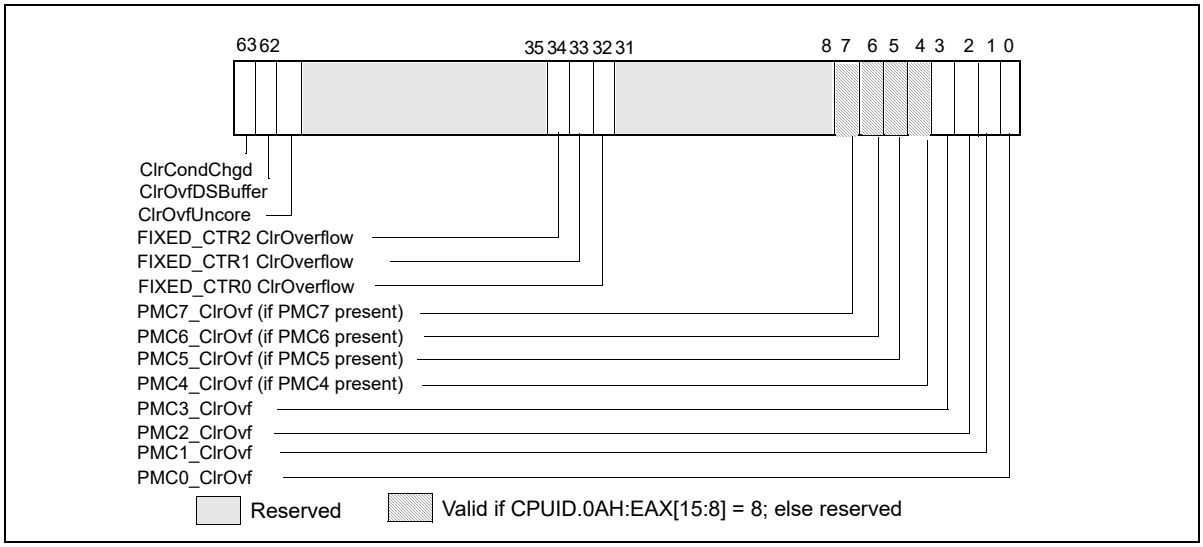


Figure 19-28. IA32_PERF_GLOBAL_OVF_CTRL MSR in Sandy Bridge Microarchitecture

19.3.4.2 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.

19.3.4.3 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 19.2.4).

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].

19.3.4.4 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 19-11.

Table 19-11. 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 19.3.1.1.1	Section 19.3.1.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 19-29	Figure 19-15	
PEBS record layout	Physical Layout same as Table 19-3.	Table 19-3	Enhanced fields at offsets 98H, A0H, A8H.
PEBS Events	See Table 19-12.	See Table 19-84.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table 19-13.	Table 19-4	
PEBS-Precise Store	Yes; see Section 19.3.4.4.3.	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.

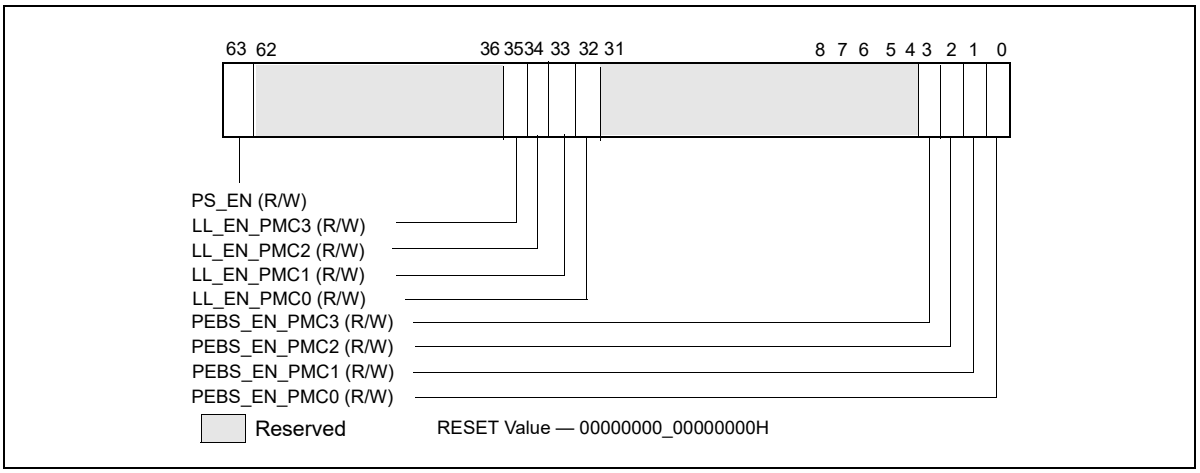


Figure 19-29. Layout of IA32_PEBS_ENABLE MSR

19.3.4.4.1 PEBS Record Format

The layout of PEBS records physically identical to those shown in Table 19-3, but the fields at offset 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 19-4. 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 19-12.

Table 19-12. 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
BR_MISP_RETIRED	C5H	Conditional	01H
		Near_Call	02H
		All_branches	04H
		Not_Taken	10H
		Taken	20H

Table 19-12. PEBS Performance Events for Sandy Bridge Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
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.

19.3.4.4.2 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 19-3 and Section 19.3.4.4.1. 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_RETIRED 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 19-3. The specificity of Data Source entry at offset A0H has been enhanced to report three pieces of information.

Table 19-13. Layout of Data Source Field of Load Latency Record

Field	Position	Description
Source	3:0	See Table 19-4
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 19-17.

19.3.4.4.3 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 offset 98H, A0H and A8H of Table 19-3. The specificity of Data Source entry at offset A0H has been enhanced to report three piece of information.

Table 19-14. 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

19.3.4.4.4 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_RETIRED 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_RETIRED 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_RETIRED.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_RETIRED.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.

19.3.4.5 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 19-15 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 19-15. 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 19-30 and Figure 19-31. 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.

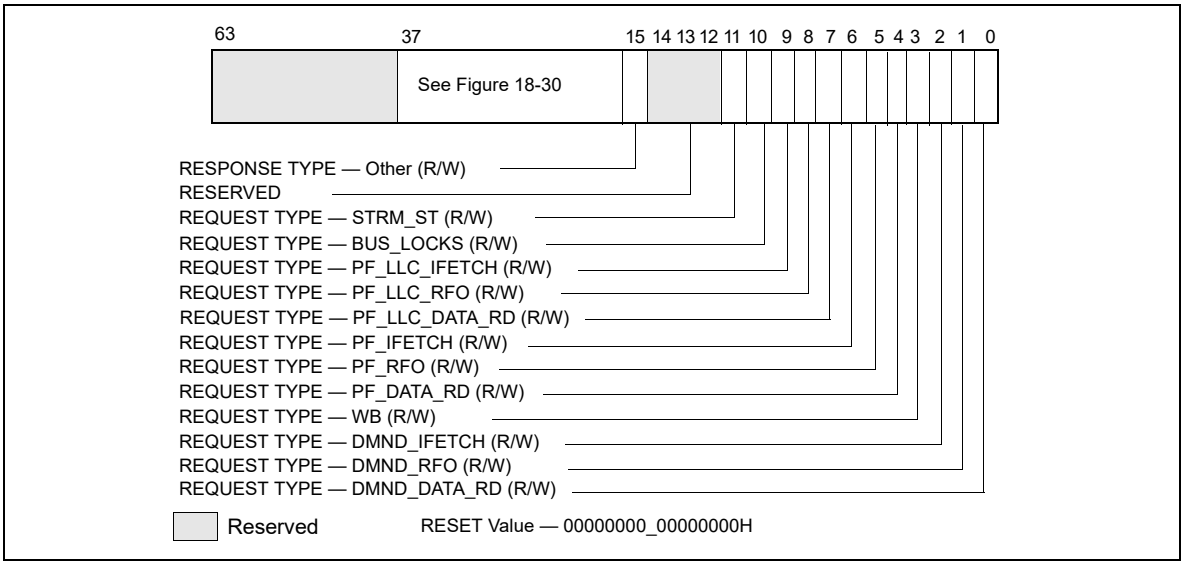


Figure 19-30. Request_Type Fields for MSR_OFFCORE_RSP_x

Table 19-16. 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.

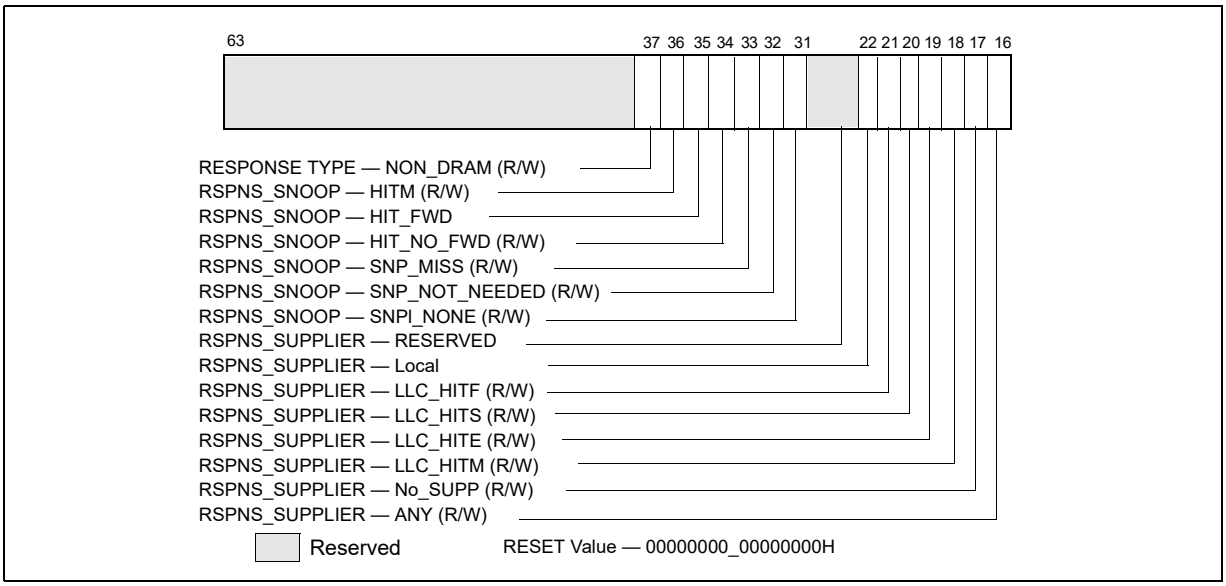


Figure 19-31. Response_Supplier and Snoop Info Fields for MSR_OFFCORE_RSP_x

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 19-17. 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 19-18. 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, ReslHitI 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.

19.3.4.6 Uncore Performance Monitoring Facilities In Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, 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 19.3.1.2.2. 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 19-32.

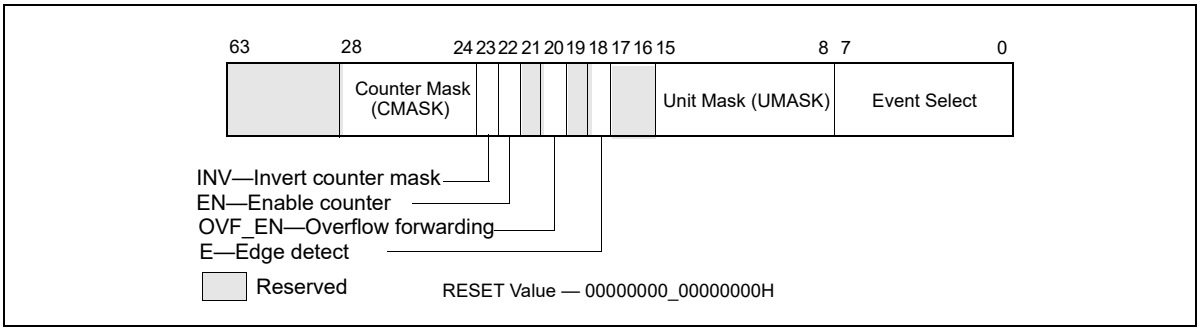


Figure 19-32. Layout of Uncore PERFEVTSEL MSR for a C-Box Unit or the ARB Unit

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 19-33 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).

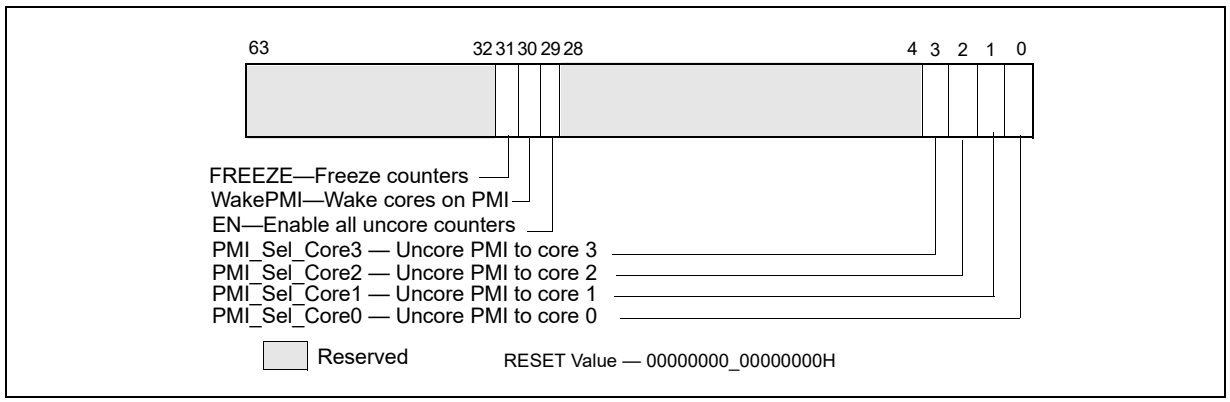


Figure 19-33. Layout of MSR_UNC_PERF_GLOBAL_CTRL MSR for Uncore

Additionally, there is also a fixed counter, counting uncore clockticks, for the uncore domain. Table 19-19 summarizes the number MSRs for uncore PMU for each box.

Table 19-19. 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 2-21 MSR_UNC_CBO_CONFIG
ARB	1	2	44	Yes	Uncore	

Table 19-19. Uncore PMU MSR Summary (Contd.)

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Comment
Fixed Counter	N.A.	N.A.	48	No	Uncore	

19.3.4.6.1 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.

19.3.4.7 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 19.3.4.8.

Thus, the performance monitoring facilities in the processor core generally are the same as those described in Section 19.6.3 through Section 19.3.4.5. However, the MSR_OFFCORE_RSP_0/MSR_OFFCORE_RSP_1 Response Supplier Info field shown in Table 19-17 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 19-20. Additionally, there are some small differences in the non-architectural performance monitoring events (see event list available at: https://perfmon-events.intel.com/).

Table 19-20. 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).

19.3.4.8 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 19-21 summarizes the uncore PMU facilities providing MSR interfaces.

Table 19-21. 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.

19.3.5 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 19.6.3 through Section 19.3.4.5. The non-architectural performance monitoring events supported by the processor core can be found at: https://perfmon-events.intel.com/.

19.3.5.1 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.

19.3.6 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 19.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 19.2.3.

The core PMU's capability is similar to those described in Section 19.6.3 through Section 19.3.4.5, with some differences and enhancements summarized in Table 19-22. 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 19.3.6.5. For details of Intel TSX, see Chapter 16, "Programming with Intel® Transactional Synchronization Extensions" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

Table 19-22. 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 19.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 19.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 19.2.2.
# 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 19.2.1.
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 17.4.7.
Processor Event Based Sampling (PEBS) Events	See Table 19-12 and Section 19.3.6.5.1.	See Table 19-12.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 19.3.4.4.2.	See Section 19.3.4.4.2.	
PEBS-Precise Store	No, replaced by Data Address profiling.	Section 19.3.4.4.3	
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 17.11.	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 19.3.6.5.	No	

19.3.6.1 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 19-23.

Table 19-23. 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 19.3.1.1.1	Section 19.3.1.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 19-15	Figure 19-29	
PEBS record layout	Table 19-24; enhanced fields at offsets 98H, A0H, A8H, B0H.	Table 19-3; enhanced fields at offsets 98H, A0H, A8H.	

Table 19-23. PEBS Facility Comparison

Box	Haswell Microarchitecture	Sandy Bridge Microarchitecture	Comment
Precise Events	See Table 19-12.	See Table 19-12.	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Table 19-13.	Table 19-13	
PEBS-Precise Store	No, replaced by data address profiling.	Yes; see Section 19.3.4.4.3.	
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.

19.3.6.2 PEBS Data Format

The PEBS record format for the 4th Generation Intel Core processor is shown in Table 19-24. 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[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 19-24. 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
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 19.3.6.5.1)

The layout of PEBS records are almost identical to those shown in Table 19-3. 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 19.3.4.4.2), PDIR (Section 19.3.4.4.4), and the equivalent capability of precise store in prior generation (see Section 19.3.6.3).

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.

19.3.6.3 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 19-25. Precise Events That Supports Data Linear Address Profiling

Event Name	Event Name
MEM_UOPS_RETIRED.STLB_MISS_LOADS	MEM_UOPS_RETIRED.STLB_MISS_STORES
MEM_UOPS_RETIRED.LOCK_LOADS	MEM_UOPS_RETIRED.SPLIT_STORES
MEM_UOPS_RETIRED.SPLIT_LOADS	MEM_UOPS_RETIRED.ALL_STORES
MEM_UOPS_RETIRED.ALL_LOADS	MEM_LOAD_UOPS_LLC_MISS_RETIRED.LOCAL_DRAM
MEM_LOAD_UOPS_RETIRED.L1_HIT	MEM_LOAD_UOPS_RETIRED.L2_HIT
MEM_LOAD_UOPS_RETIRED.L3_HIT	MEM_LOAD_UOPS_RETIRED.L1_MISS
MEM_LOAD_UOPS_RETIRED.L2_MISS	MEM_LOAD_UOPS_RETIRED.L3_MISS
MEM_LOAD_UOPS_RETIRED.HIT_LFB	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_MISS
MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HIT	MEM_LOAD_UOPS_L3_HIT_RETIRED.XSNP_HITM
UOPS_RETIRED.ALL (if load or store is tagged)	MEM_LOAD_UOPS_LLC_HIT_RETIRED.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 19-25 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 19-26.

Table 19-26. 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 19-25.
Reserved	A8H	Always zero.

19.3.6.3.1 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.

19.3.6.4 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 19.3.4.5. The event codes are listed in Table 19-15. 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 19-27.
- Supplier information (bits 30:16): see Table 19-28.
- Snoop response information (bits 37:31): see Table 19-18.

Table 19-27. 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.
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

Table 19-27. MSR_OFFCORE_RSP_x Request_Type Definition (Haswell Microarchitecture) (Contd.)

Bit Name	Offset	Description
OTHER	15	Any other request that crosses IDI, including I/O.

The supplier information field listed in Table 19-28. The fields vary across products (according to CPUID signatures) and is noted in the description.

Table 19-28. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signature 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 19-29. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID 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

19.3.6.4.1 Off-core Response Performance Monitoring in Intel Xeon Processors E5 v3 Series

Table 19-28 lists the supplier information field that apply to Intel Xeon processor E5 v3 series (CPUID signature 06_3FH).

Table 19-30. 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

19.3.6.5 Performance Monitoring and Intel® TSX

Chapter 16 of *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.(EAX=7, ECX=0):RTM[bit 11]=1, or (b) if CPUID.07H.EBX.HLE [bit 4] = 1. The TSX-enhanced layout of IA32_PERFEVTSELx is shown in Figure 19-34. 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.

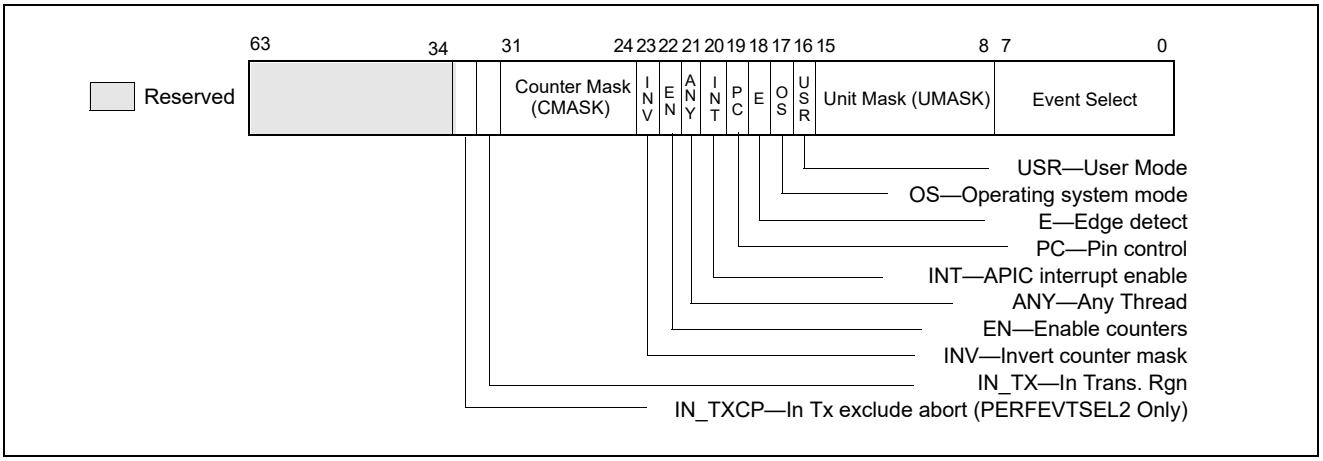


Figure 19-34. Layout of IA32_PERFEVTSELx MSRs Supporting Intel TSX

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/.

19.3.6.5.1 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 19-31.

Table 19-31. 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.
Reserved	63:40	Reserved

19.3.6.6 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 19.3.4.6.

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 19-32.

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 19-33 shows the layout of the uncore domain global control.

Additionally, there is also a fixed counter, counting uncore clockticks, for the uncore domain. Table 19-19 summarizes the number MSRs for uncore PMU for each box.

Table 19-32. 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 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/.

19.3.6.7 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.

19.3.7 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 19.2.3) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 3 capabilities are described in Section 19.2.3.

The core PMU has the same capability as those described in Section 19.3.6. 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.

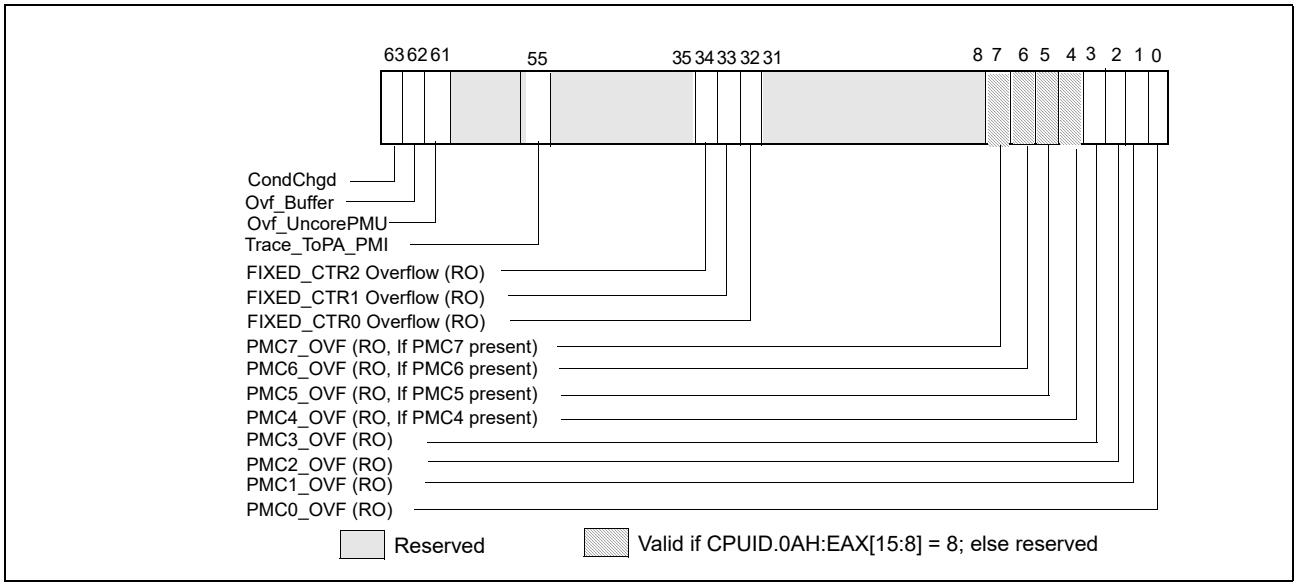


Figure 19-35. IA32_PERF_GLOBAL_STATUS MSR in Broadwell Microarchitecture

Details of Intel Processor Trace is described in Chapter 32, "Intel® Processor Trace". The IA32_PERF_GLOBAL_OVF_CTRL MSR provides a corresponding reset control bit.

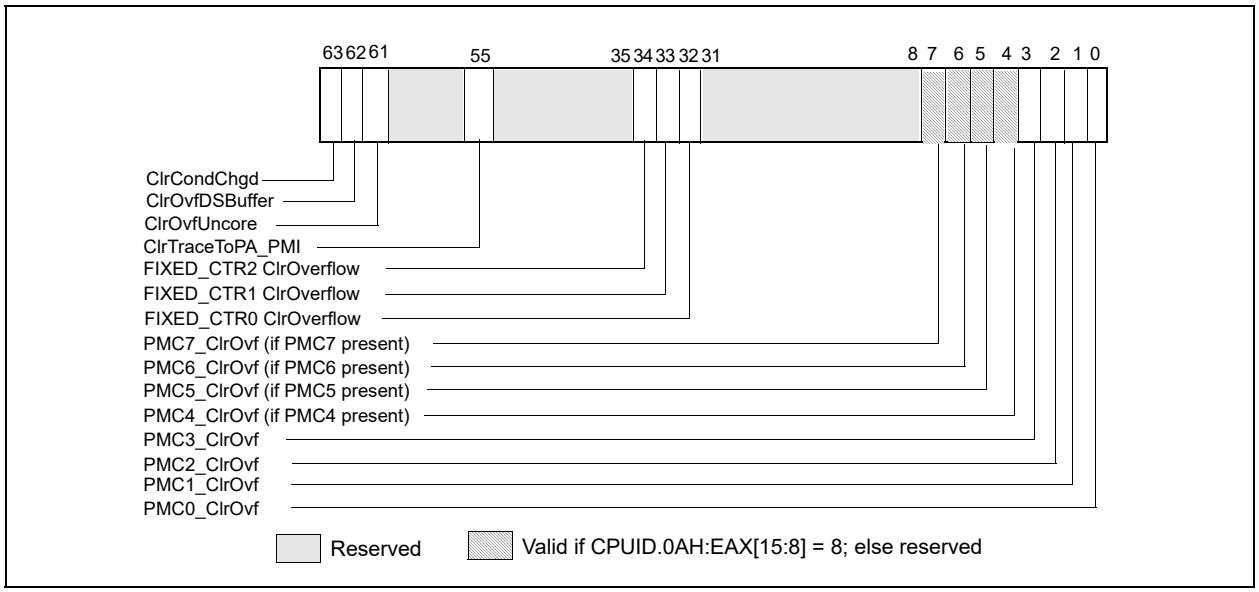


Figure 19-36. IA32_PERF_GLOBAL_OVF_CTRL MSR in Broadwell microarchitecture

The specifics of non-architectural performance events can be found at: https://perfmon-events.intel.com/.

19.3.8 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 19.2.4) and a host of non-architectural monitoring capabilities.

Architectural performance monitoring version 4 capabilities are described in Section 19.2.4.

The core PMU's capability is similar to those described in Section 19.6.3 through Section 19.3.4.5, with some differences and enhancements summarized in Table 19-33. 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 19.3.6.5. For details of Intel TSX, see Chapter 16, "Programming with Intel® Transactional Synchronization Extensions" 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 39, "Enclave Code Debug and Profiling".

Table 19-33. 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 19.2.1.
# of general-purpose counters per core	8	8	Use CPUID to determine # of counters. See Section 19.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	See Section 19.2.2.
# 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 19.2.1.
Architectural Perfmon version	4	3	See Section 19.2.4
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 17.4.7. 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 19.2.4.

Table 19-33. 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 19.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	▪ CTR_Frz ▪ LBR_Frz	NA	See Section 19.2.4.1.
Perfmon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	NA	See Section 19.2.4.3.
Precise Events	See Table 19-36.	See Table 19-12.	IA32_PMC4-PMC7 do not support PEBS.
PEBS for front end events	See Section 19.3.8.2.	No	
LBR Record Format Encoding	000101b	000100b	Section 17.4.8.1
LBR Size	32 entries	16 entries	
LBR Entry	From_IP/To_IP/LBR_Info triplet	From_IP/To_IP pair	Section 17.12
LBR Timing	Yes	No	Section 17.12.1
Call Stack Profiling	Yes, see Section 17.11	Yes, see Section 17.11	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 19.3.6.5.	See Section 19.3.6.5.	

19.3.8.1 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 19-34.

Table 19-34. 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 19.3.1.1.1	Section 19.3.1.1.1	Unchanged
IA32_PEBS_ENABLE Layout	Figure 19-15	Figure 19-15	
PEBS-EventingIP	Yes	Yes	
PEBS record format encoding	0011b	0010b	
PEBS record layout	Table 19-35; enhanced fields at offsets 98H- B8H; and TSC record field at C0H.	Table 19-24; enhanced fields at offsets 98H, A0H, A8H, B0H.	
Multi-counter PEBS resolution	PEBS record 90H resolves the eventing counter overflow.	PEBS record 90H reflects IA32_PERF_GLOBAL_STATUS.	
Precise Events	See Table 19-36.	See Table 19-12.	IA32_PMC4-IA32_PMC7 do not support PEBS.

Table 19-34. PEBS Facility Comparison (Contd.)

Box	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Haswell and Broadwell Microarchitectures	Comment
PEBS-PDIR	Yes	Yes	IA32_PMC1 only.
PEBS-Load Latency	See Section 19.3.4.4.2.	See Section 19.3.4.4.2.	
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

Precise 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.

19.3.8.1.1 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 19-35. 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[bit 2] reports whether the processor's DS storage format support is mode-independent. When set, it uses 64-bit DS storage format.

Table 19-35. PEBS Record Format for 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)
48H	R/ESP	B0H	EventingIP
50H	R8	B8H	TX Abort Information (Section 19.3.6.5.1)
58H	R9	C0H	TSC
60H	R10		

The layout of PEBS records are largely identical to those shown in Table 19-24.

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 19.3.4.4.2), PDIR (Section 19.3.4.4.4), and data address profiling (Section 19.3.6.3).

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.

19.3.8.1.2 PEBS Events

The list of precise events supported for PEBS in the Skylake, Kaby Lake and Coffee Lake microarchitectures is shown in Table 19-36.

Table 19-36. 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
MEM_INST_RETIRED[2]	D0H	LOCK_LOADS	21H
		SPLIT_LOADS	41H
		SPLIT_STORES	42H
		ALL_LOADS	81H
		ALL_STORES	82H

Table 19-36. Precise Events for the Skylake, Kaby Lake and Coffee Lake Microarchitectures (Contd.)

Event Name	Event Select	Sub-event	UMask
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_PEBS_FRONTEND, see Section 19.3.8.3
4. Instruction with at least one load uop experiencing the condition specified in the UMask.

19.3.8.1.3 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 19-26.

Table 19-37. 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.

19.3.8.2 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 retirement (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 19-38.
- 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 19-38.

Table 19-38. 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 19-39.

Table 19-39. 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 19-38.
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.

19.3.8.3 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 19.3.4.5. 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 19-40.
- Supplier information (bits 29:16): see Table 19-41.
- Snoop response information (bits 37:30): see Table 19-42.

Table 19-40. 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 un-cacheable accesses.

Table 19-41 lists the supplier information field that applies to 6th generation, 7th generation and 8th generation Intel Core processors. (6th generation Intel Core processor CPUID signatures: 06_4EH, 06_5EH; 7th generation and 8th generation Intel Core processor CPUID signatures: 06_8EH, 06_9EH).

Table 19-41. MSR_OFFCORE_RSP_x Supplier Info Field Definition (CPUID Signatures 06_4EH, 06_5EH and 06_8EH, 06_9EH)

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	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 19-42 lists the snoop information field that apply to processors with CPUID signatures 06_4EH, 06_5EH, 06_8EH, 06_9E, and 06_55H.

Table 19-42. MSR_OFFCORE_RSP_x Snoop Info Field Definition (CPUID Signatures 06_4EH, 06_5EH, 06_8EH, 06_9E and 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, ReslHitI was returned by all cores. -For LLC Miss, RspI 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.

19.3.8.3.1 Off-core Response Performance Monitoring for the Intel® Xeon® Processor Scalable Family

The following tables list the requestor and supplier information fields that apply to the Intel® Xeon® Processor Scalable Family.

- Transaction request type encoding (bits 15:0): see Table 19-43.
- Supplier information (bits 29:16): see Table 19-44.
- Supplier information (bits 29:16) with support for Intel® Optane™ DC Persistent Memory support: see Table 19-45.
- Snoop response information has not been changed and is the same as in (bits 37:30): see Table 19-42.

Table 19-43. MSR_OFFCORE_RSP_x Request_Type Definition (Intel® Xeon® Processor Scalable 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 19-44 lists the supplier information field that applies to the Intel Xeon Processor Scalable Family (CPUID signature: 06_55H).

Table 19-44. 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 19-45 lists the supplier information field that applies to the Intel Xeon Processor Scalable Family (CPUID signature: 06_55H, Steppings 0x5H - 0xFH).

Table 19-45. 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

19.3.8.4 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 performance 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*.

19.3.9 10th Generation Intel® Core™ Processor Performance Monitoring Facility

Some 10th generation Intel® Core™ processors and some 3rd generation Intel® Xeon® Processor Scalable 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 19.2.5) and a host of non-architectural monitoring capabilities.

The core PMU's capability is similar to those described in Section 19.3.1 through Section 19.3.8, with some differences and enhancements summarized in Table 19-46.

Table 19-46. 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 19.2.5.
PEBS: Basic functionality	Yes	Yes	See Section 19.3.9.1.

Table 19-46. Core PMU Summary of the Ice Lake Microarchitecture

Box	Ice Lake Microarchitecture	Skylake, Kaby Lake and Coffee Lake Microarchitectures	Comment
PEBS record format encoding	0100b	0011b	See Section 19.6.2.4.2.
Extended PEBS	PEBS is extended to all Fixed and General Purpose counters and to all performance monitoring events.	No	See Section 19.9.1.
Adaptive PEBS	Yes	No	See Section 19.9.2.
Performance Metrics	Yes (4)	No	See Section 19.3.9.3.
PEBS-PDIR	IA32_FIXED0 only (Corresponding counter control MSRs must be enabled.)	IA32_PMC1 only.	

19.3.9.1 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 19.9.

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 19.9.5 for details.

19.3.9.2 Off-core Response Performance Monitoring

The core PMU facility to collect off-core response events are similar to those described in Section 19.3.4.5. 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 19-47. 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.

Table 19-47. MSR_OFFCORE_RSP_x Request_Type Definition (Processors Based on Ice Lake Microarchitecture)

Bit Name	Offset	Description
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 19-48 lists the supplier information field that applies to processors based on Ice Lake microarchitecture.

Table 19-48. 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.
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 19-49 lists the snoop information field that applies to processors based on Ice Lake microarchitecture.

Table 19-49. 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.

19.3.9.3 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*.

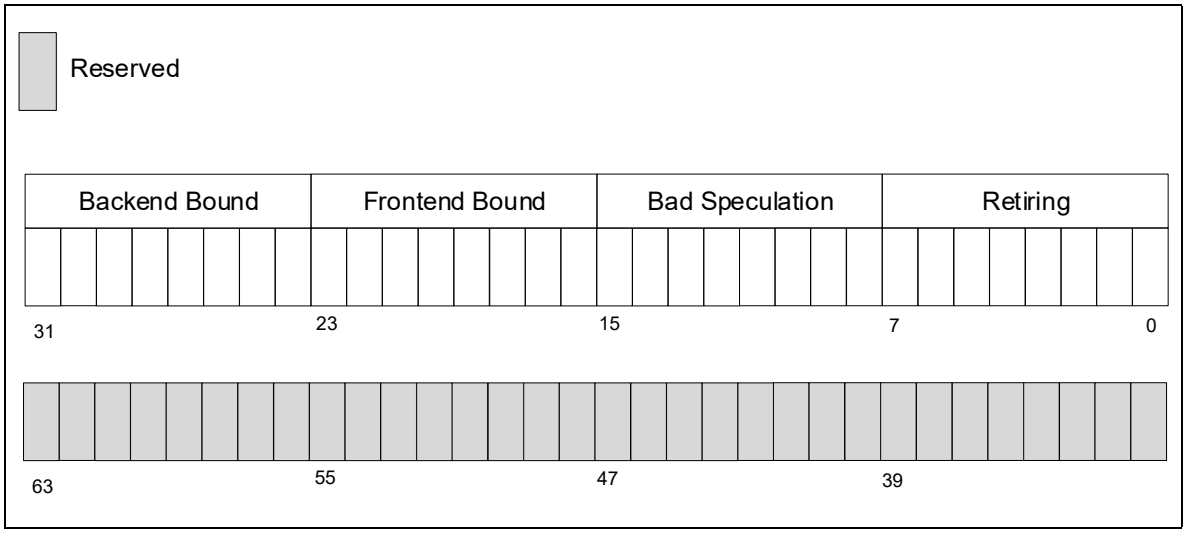


Figure 19-37. MSR_PERF_METRICS Definition

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 triggered[1]. 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].
- On state restore, fixed-function performance-monitoring counter 3 must be restored **before** PERF_METRICS, otherwise undefined results may be observed.

19.3.10 12th Generation Intel® Core™ Processor 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. They report architectural performance monitoring version ID = 5 and support non-architectural monitoring capabilities described in this section.

19.3.10.1 P-core Performance Monitoring Unit

The P-core PMU's capability is similar to those described in Section 19.3.1 through Section 19.3.9, with some differences and enhancements summarized in Table 19-50.

1. An overflow of fixed-function performance-monitoring counter 3 should normally happen first if software follows Intel's recommendations.

Table 19-50. Core PMU Summary of the Golden Cove Microarchitecture

Box	Golden Cove Microarchitecture	Ice Lake Microarchitecture	Comment
Architectural Perfmon version	5	5	See Section 19.2.5.
Event-Counter Restrictions	Simplified identification		Counters 4-7 support a subset of events. See Section 19.3.10.1.2.
Performance Metrics	Yes (12)	Yes (4)	See Section 19.3.9.3.
PEBS: Baseline, record format	Yes 0100b	Yes 0100b	See Section 19.3.9.
PEBS: EPT-friendly	Yes	No; debuts in Ice Lake server microarchitecture	See Section 19.6.2.4.2.
PEBS: Precise Distribution	IA32_FIXED0 instruction-granularity PDist on IA32_PMC0	IA32_FIXED0 cycle-granularity No PDist	See Section 19.9.6.
PEBS: Load Latency	Instruction latency Cache latency Access info fields (5)	Instruction latency Access info fields (3)	See Section 19.9.7.
PEBS: Store Latency	Cache latency Access info fields (3)	None	See Section 19.9.8.

19.3.10.1.1 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 19-38.

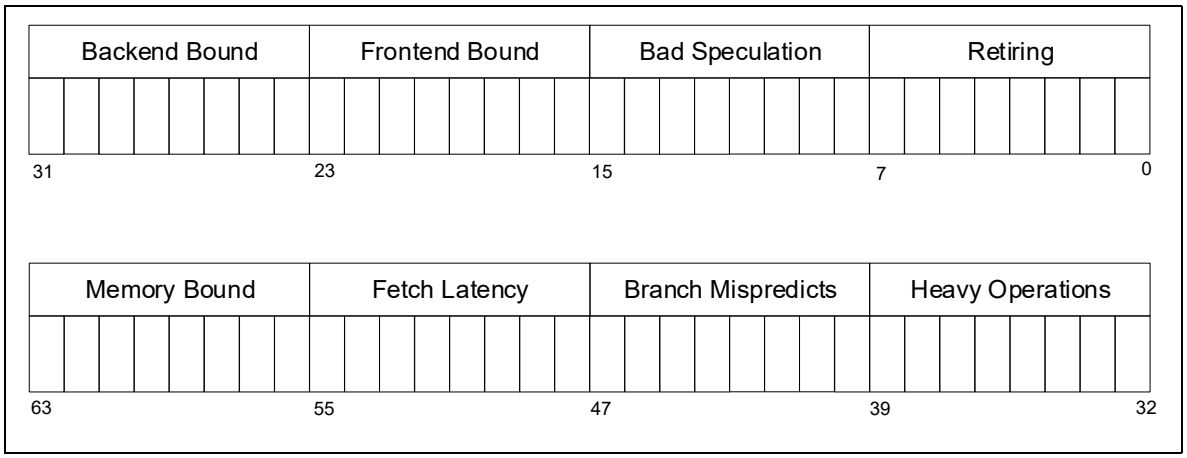


Figure 19-38. PERF_METRICS MSR Definition for 12th Generation Intel® Core™ Processor P-core

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 19-39.

```
Light_Operations = Retiring - Heavy_Operations
Machine_Clears = Bad_Speculation - Branch_Mispredicts
Fetch_Bandwidth = Frontend_Bound - Fetch_Latency
Core_Bound = Backend_Bound - Memory_Bound
```

Figure 19-39. Deducing Implied Level 2 Metrics in the Core PMU for12th 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.

19.3.10.1.2 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 19-51 lists the exceptions to this rule.

Table 19-51. 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
04A4	TOPDOWN.BAD_SPEC_SLOTS	0
08A4	TOPDOWN.BR_MISPREDICT_SLOTS	
xxCE	AMX_OPS_RETIRED	0

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_PERFEVTSELx MSRs.

19.3.10.1.3 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 19.3.9.2.

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.

19.3.10.2 E-core Performance Monitoring Unit

The core PMU capabilities on the 12th generation Intel Core processor E-core are summarized in Table 19-52 below.

Table 19-52. 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 19.2.1.
Number of general-purpose counters per core	6	4	Use CPUID to enumerate number of counters. See Section 19.2.1.
Architectural Performance Monitoring version ID	5	5	See Section 19.2.5.
PEBS record format encoding	0100b	0100b	See Section 19.5.5.
EPT-friendly PEBS support	Yes	No	See Section 19.9.5.
Extended PEBS	Yes	Yes	See Section 19.9.1.
Adaptive PEBS	Yes	Yes	See Section 19.9.2.
Precise distribution (PDist) PEBS	IA32_PMC0 and IA32_FIXED_CTR0	IA32_PMC0 and IA32_FIXED_CTR0	PDist eliminates skid, see Section 19.9.3, Section 19.9.4, and Section 19.9.6.
PEBS Latency	Load and Store Latency	No	See Section 19.3.10.2.1, Section 19.3.10.2.2, Section 19.9.7, and Section 19.9.8.
PEBS Output	DS Save Area or Intel® Processor Trace	DS Save Area or Intel® Processor Trace	See Section 19.5.5.2.1.
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 19.5.5.4.

19.3.10.2.1 E-core PEBS Load Latency

The 12th generation Intel Core processor E-core includes PEBS Load Latency support similar to that described in Section 19.9.7.

When a programmable counter is configured to count MEM_UOPS_RETIRED.LOAD_LATENCY_ABOVE_THRESHOLD (IA32_PERFEVTSELx[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 19.9.2.2.2). The formats of these fields are shown in Table 19-53 and Table 19-94.

Table 19-53. E-core PEBS Memory Access Info Encoding

Bit(s)	Field	Description
3:0	Data Source	The source of the data; see Table 19-54.
4	Lock	0: The operation was not part of a locked transaction. 1: The operation was part of a locked transaction.

Table 19-53. E-core PEBS Memory Access Info Encoding (Contd.)

Bit(s)	Field	Description
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 19-94.

Table 19-54. E-core PEBS Data Source Encodings

Encoding	Description
00H	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.
0AH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to shared state).
0BH	Reserved.
0CH	L3 MISS. Local home requests that missed the L3 cache and were serviced by local DRAM (go to exclusive state).
0DH	Reserved.
0EH	I/O. Request of input/output operation.
0FH	The request was to un-cacheable memory.

19.3.10.2.2 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_PERFEVTSELx[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 18.9.2.2.2). The formats of these fields are shown in Table 19-53 and Table 19-94.

19.3.10.2.3 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 19.9.6 for additional details on PDist.

19.3.10.2.4 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 19-55.
- Bits 30:16 specify Response Type information or an L2 Hit, and is described in Table 19-75.
- If L2 misses, then bits 37:31 can be used to specify snoop response information and is described in Table 19-76.
- 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 19.5.2.3 for details.

Table 19-55. 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 prefteches 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).
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.

19.3.10.3 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 leaf 0AH.EBX (unlike prior hybrid parts).

19.4 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.

19.4.1 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 19.2.3) 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 19-6 and described in Section 19.2.1.1 and Section 19.2.3. The processor supports AnyThread counting in three architectural performance monitoring events.

19.4.1.1 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.

19.4.1.1.1 Processor Event-Based Sampling

The processor supports processor event based sampling (PEBS). PEBS is supported using IA32_PMC0 (see also Section 17.4.9, “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 19.6.2.4).

The list of PEBS events supported in the processor is shown in the following table.

Table 19-56. PEBS Performance Events for the 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
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 19-57, and each field in the PEBS record is 64 bits long.

Table 19-57. PEBS Record Format for the 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

Table 19-57. PEBS Record Format for the Knights Landing Microarchitecture (Contd.)

Byte Offset	Field	Byte Offset	Field
58H	R9	B8H	Reserved

19.4.1.1.2 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 19-58 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 19-58. 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 19-59. 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 19.5.2.3 for details.

Table 19-59. 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.

Table 19-59. Bit fields of the MSR_OFFCORE_RESP [0, 1] Registers (Contd.)

Main	Sub-field	Bit	Name	Description
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.
	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 PMC0. 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 PMC0. This bit is reserved for OFFCORE_RESP_1 event.

19.4.1.1.3 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 19.5.2.3, "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.

19.5 PERFORMANCE MONITORING (INTEL ATOM® PROCESSORS)

19.5.1 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 19.2.3) 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 19.2.1.

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 19-6 and described in Section 19.2.1.1 and Section 19.2.3.

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 19-77, Table 19-78, Table 19-79, and Table 19-80. 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 17.4.9, “BTS and DS Save Area”).

19.5.2 Performance Monitoring for Silvermont Microarchitecture

Intel processors based on the Silvermont microarchitecture report architectural performance monitoring versionID = 3 (see Section 19.2.3) 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 19-6 and described in Section 19.2.1.1 and Section 19.2.3. Architectural and non-architectural performance monitoring events in the Silvermont microarchitecture ignore the AnyThread qualification regardless of its setting in IA32_PERFEVTSELx MSR.

19.5.2.1 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.

19.5.2.1.1 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 17.4.9).

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 19.6.2.4).

The list of precise events supported in the Silvermont microarchitecture is shown in Table 19-60.

Table 19-60. 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 19-61, and each field in the PEBS record is 64 bits long.

Table 19-61. 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

19.5.2.2 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 19-62 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 19-62. 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)

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are shown in Figure 19-40 and Figure 19-41. 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 19.5.2.3 for details.

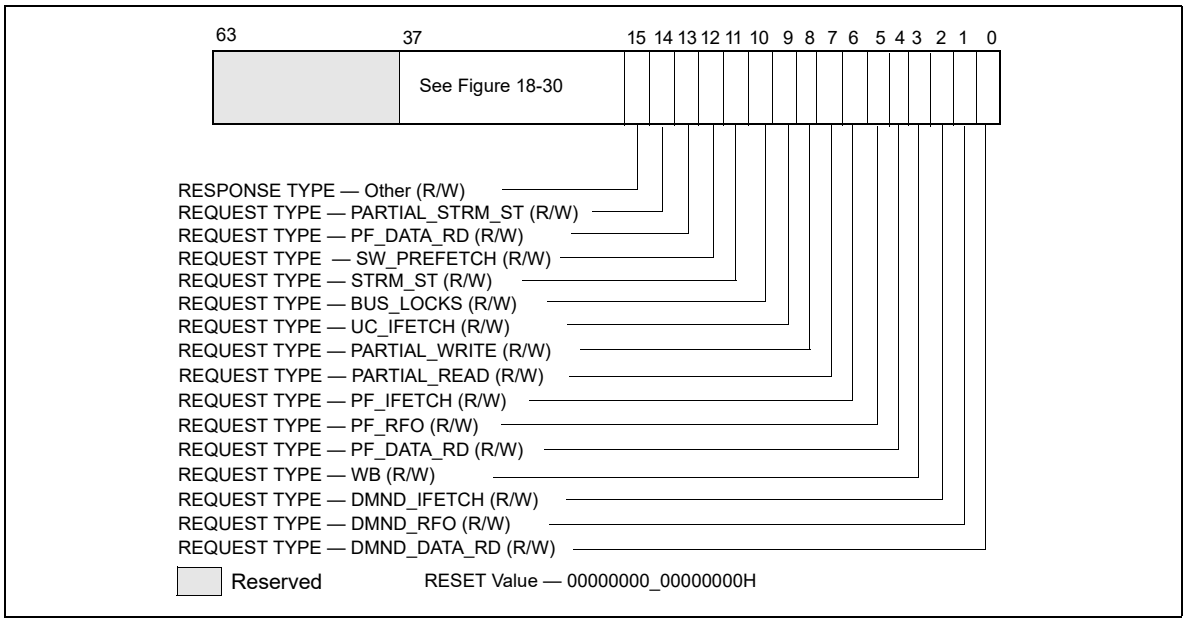


Figure 19-40. Request_Type Fields for MSR_OFFCORE_RSPx

Table 19-63. 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.

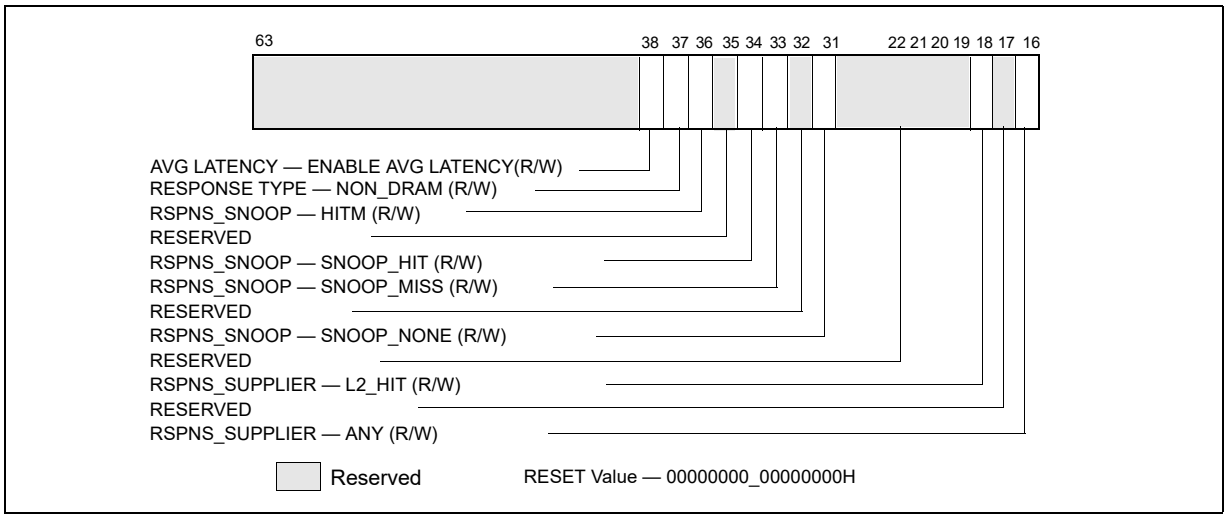


Figure 19-41. Response_Supplier and Snoop Info Fields for MSR_OFFCORE_RSPx

To properly program this extra register, software must set at least one request type bit (Table 19-63) and a valid response type pattern (Table 19-64, Table 19-65). 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 19-64. 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 19-65. 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.
	Reserved	35	Reserved

Table 19-65. MSR_OFFCORE_RSPx Snoop Info Field Definition (Contd.)

Subtype	Bit Name	Offset	Description
	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.

19.5.2.3 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_PERFEVTSELx 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, white 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.

19.5.3 Performance Monitoring for Goldmont Microarchitecture

Intel Atom processors based on the Goldmont microarchitecture report architectural performance monitoring versionID = 4 (see Section 19.2.4) and support non-architectural monitoring capabilities described in this section.

Architectural performance monitoring version 4 capabilities are described in Section 19.2.4.

The bit fields (except bit 21) within each IA32_PERFEVTSELx MSR are defined in Figure 19-6 and described in Section 19.2.1.1 and Section 19.2.3. 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_PERFEVTSELx 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 19.5.2 , with some differences and enhancements summarized in Table 19-66.

Table 19-66. 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 19.2.1.
# of general-purpose counters per core	4	2	Use CPUID to determine # of counters. See Section 19.2.1.
Counter width (R,W)	R:48, W: 32/48	R:40, W:32	See Section 19.2.2.
Architectural Performance Monitoring version ID	4	3	Use CPUID to determine # of counters. See Section 19.2.1.

Table 19-66. 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 17.4.7. 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 19.2.4.
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 19.2.4.
Enable control in IA32_PERF_GLOBAL_STATUS	▪ CTR_Frz, ▪ LBR_Frz	No	See Section 19.2.4.1.
Perfmon Counter In-Use Indicator	Query IA32_PERF_GLOBAL_INUSE	No	See Section 19.2.4.3.
Processor Event Based Sampling (PEBS) Events	General-Purpose Counter 0 only. Supports all events (precise and non-precise). Precise events are listed in Table 19-67.	See Section 19.5.2.1.1. General-Purpose Counter 0 only. Only supports precise events (see Table 19-60).	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 19-68; enhanced fields at offsets 90H- 98H; and TSC record field at C0H.	Table 19-61.	
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.

19.5.3.1 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 17.4.9).

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 19.6.2.4 and Section 17.4.9). 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 19-67.

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 over-flows, 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 19-67. 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_RETITRED	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
BR_MISP_RETIRED	C5H	ALL_BRANCHES	00H
		JCC	7EH
		TAKEN_JCC	FEH
		IND_CALL	FBH
		NON_RETURN_IND	EBH
		RETURN	F7H

Table 19-67. Precise Events Supported by the Goldmont Microarchitecture (Contd.)

Event Name	Event Select	Sub-event	UMask
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 19-68, and each field in the PEBS record is 64 bits long.

Table 19-68. 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.

19.5.3.1.1 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.

19.5.3.1.2 Reduced Skid PEBS

Processors based on Goldmont Plus microarchitecture support the Reduced Skid PEBS feature described in Section 19.9.4 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.

19.5.3.1.3 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_Theshold, 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.

19.5.3.2 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 19-62 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 19-69.
- Bits 30:16 specifies common supplier information or an L2 Hit, and is described in Table 19-64.
- If L2 misses, then Bits 37:31 can be used to specify snoop response information and is described in Table 19-70.
- 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 19.5.2.3 for details.

Table 19-69. 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).
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.

Table 19-69. MSR_OFFCORE_RSPx Request_Type Field Definition (Contd.)

Bit Name	Offset	Description
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 19-63) and a valid response type pattern (either Table 19-64 or Table 19-70). 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 19-70. 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_OR_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_CORE_NO_FWD	34	A snoop hit in the other processor module, but no data forwarding is required.
	Reserved	35	Reserved
	L2_MISS.HITM_OTHER_CORE	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 19.5.2.3, "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

19.5.3.3 Average Offcore Request Latency Measurement

In Goldmont microarchitecture, measurement of average latency of offcore transaction requests is the same as described in Section 19.5.2.3.

19.5.4 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 19.2.4.

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_PERFEVTSELx and IA32_FIXED_CTR_CTRL have no effect.

The core PMU's capability is similar to that of the Goldmont microarchitecture described in Section 19.6.3, with some differences and enhancements summarized in Table 19-71.

Table 19-71. 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 19.2.1.
# of general-purpose counters per core	4	4	Use CPUID to determine # of counters. See Section 19.2.1.
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 19-67.	Goldmont Plus supports PEBS on all counters.
PEBS record format encoding	0011b	0011b	No change.

19.5.4.1 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.

19.5.5 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 19.2.5.

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 19-72.

Table 19-72. 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 19.2.1.
# of general-purpose counters per core	4	4	Use CPUID to determine # of counters. See Section 19.2.1.
Counter width (R,W)	R:48, W: 32/48	R:48, W: 32/48	No change. See Section 19.2.2.
Architectural Performance Monitoring version ID	5	4	
PEBS record format encoding	0100b	0011b	See Section 19.6.2.4.2.
Reduce skid PEBS	IA32_PMC0 and IA32_FIXED_CTR0	IA32_PMC0 only	
Extended PEBS	Yes	Yes	See Section 19.5.4.1.
Adaptive PEBS	Yes	No	See Section 19.9.2.
PEBS output	DS Save Area or Intel® Processor Trace	DS Save Area only	See Section 19.5.5.2.1.
PEBS record layout	See Section 19.9.2.3 for output to DS, Section 19.5.5.2.2 for output to Intel PT.	Table 19-68; enhanced fields at offsets 90H- 98H; and TSC record field at C0H.	
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.	

19.5.5.1 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 19.9.2.

19.5.5.2 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.0.EBX[25]=1, and set the new IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16] bit, support these extensions.

19.5.5.2.1 PEBS Configuration

PEBS output to Intel Processor Trace includes support for two new fields in IA32_PEBS_ENABLE.

Table 19-73. 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 32.2.7.

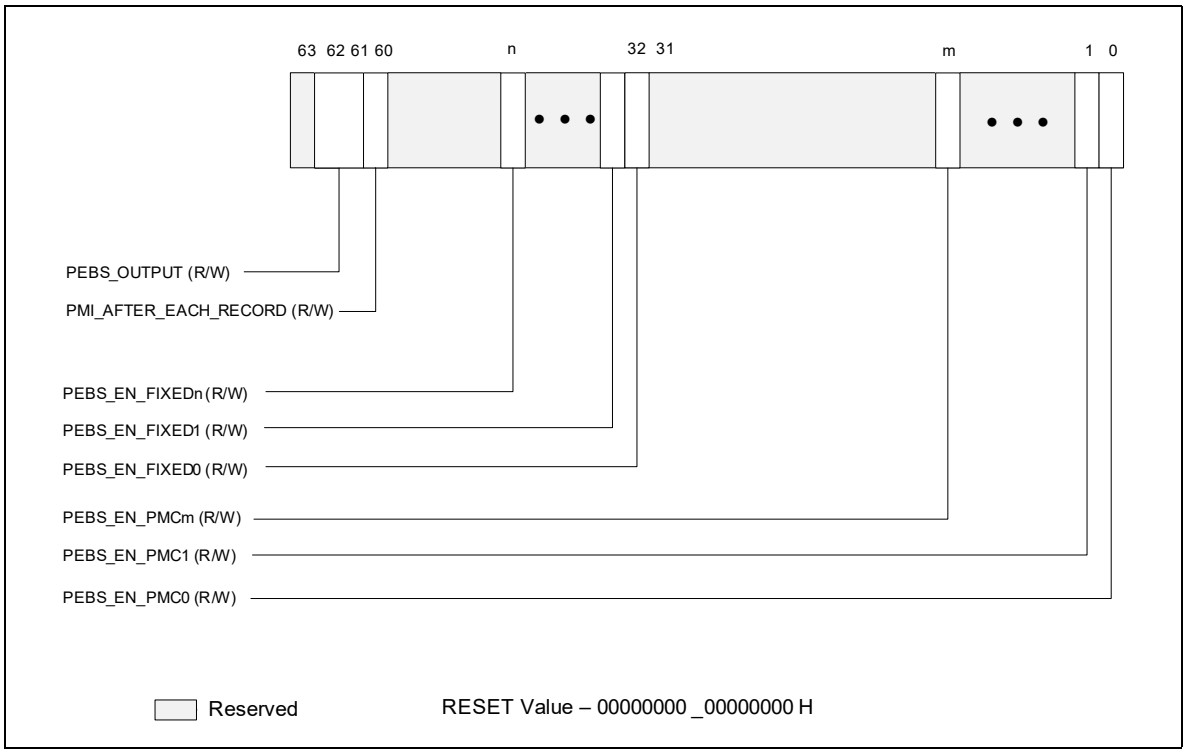


Figure 19-42. IA32_PEBS_ENABLE MSR with PEBS Output to Intel® Processor Trace

19.5.5.2.2 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 32.4.1.1 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 32.2.7.2 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.

19.5.5.2.3 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.

19.5.5.3 Precise Distribution Support on Fixed Counter 0

The Tremont microarchitecture supports the PDIR (Precise Distribution of Retired Instructions) facility, as described in Section 19.3.4.4.4, 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 19.5.3.1.2.

19.5.5.4 Compatibility Enhancements to Offcore Response MSRs

The Off-core Response facility is similar to that described in Section 19.5.3.2.

The layout of MSR_OFFCORE_RSP0 and MSR_OFFCORE_RSP1 are organized as shown below. RequestType bits are defined in Table 19-74, ResponseType bits in Table 19-75, and SnoopInfo bits in Table 19-76.

Table 19-74. 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.
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).

Table 19-74. MSR_OFFCORE_RSPx Request Type Definition (Contd.)

Bit Name	Offset	Description
L1WB_M	48	Counts modified WriteBacks from L1 that miss the L2.
L2WB_M	49	Counts modified WriteBacks from L2.

Table 19-75. 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 19-76. 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.
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 19-74.
- 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 19.5.2.3 for further details.

19.6 PERFORMANCE MONITORING (LEGACY INTEL PROCESSORS)

19.6.1 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 19-1) 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 8.6, "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 19-77. 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 19-77. 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 19-78.

Table 19-78. 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 19-79.

Table 19-79. 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 19-80. 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 19-80. 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
Bit 9	Counts shared state
Bit 8	Counts Invalid state

19.6.2 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_PERFEVTSELx MSRs for detailed event configurations), or fixed-function performance counters (see Section 19.6.2.1). IA32_PERFEVTSELx MSRs are architectural; their layout is shown in Figure 19-1. 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 19.2.2).

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 19-77, Table 19-78, Table 19-79, and Table 19-80. One or more of these sub-fields may apply to specific events on an event-by-event basis.

In addition, the UMASK filed 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 19-81.

Table 19-81. Bus Snoop Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx 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 19-82.

Table 19-82. Snoop Type Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx 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.

19.6.2.1 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 19-2 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 19-43. Two sub-fields are defined for each control. See Figure 19-43; 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.

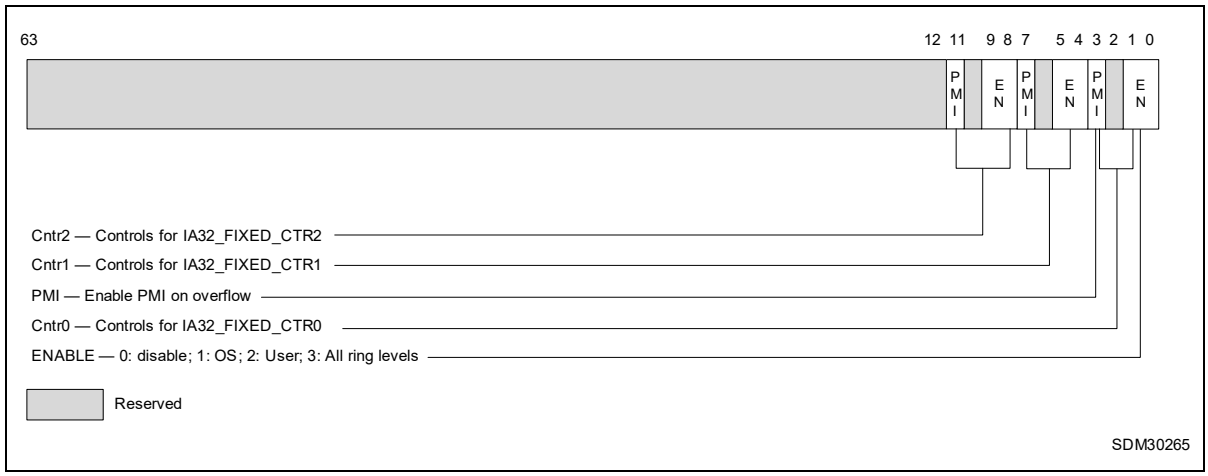


Figure 19-43. Layout of IA32_FIXED_CTR_CTRL MSR

- **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.

19.6.2.2 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 19-44). Each enable bit in MSR_PERF_GLOBAL_CTRL is AND'ed 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 AND'ed results is true; counting is disabled when the result is false.

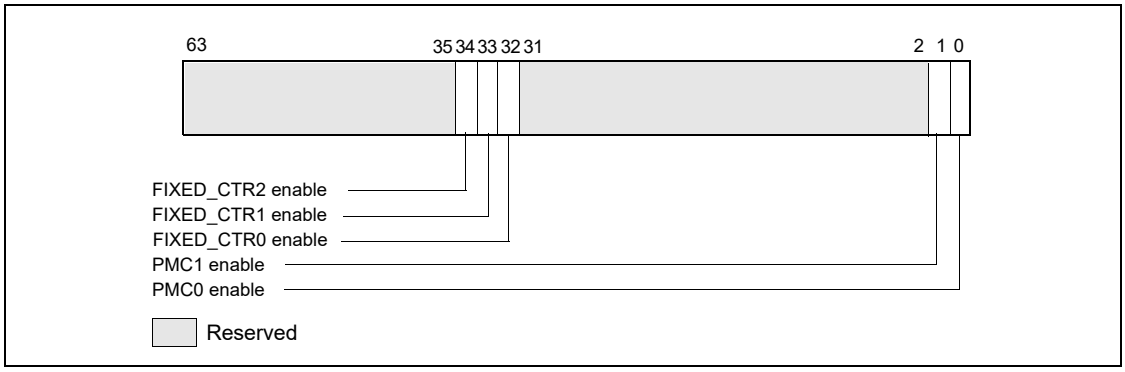


Figure 19-44. Layout of MSR_PERF_GLOBAL_CTRL MSR

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 19-45). A value of 1 in bits 34:32, 1, 0 indicates an overflow condition has occurred in the associated counter.

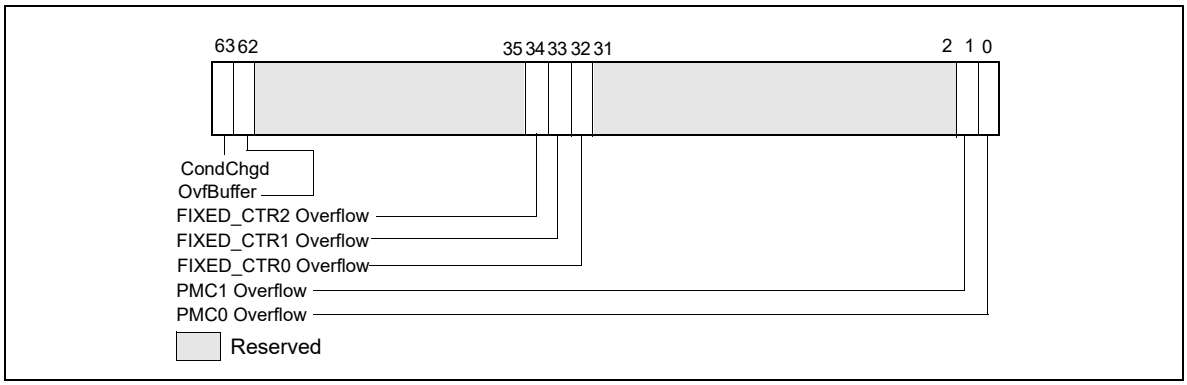


Figure 19-45. Layout of MSR_PERF_GLOBAL_STATUS MSR

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 17.4.9). 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 19-46). 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.

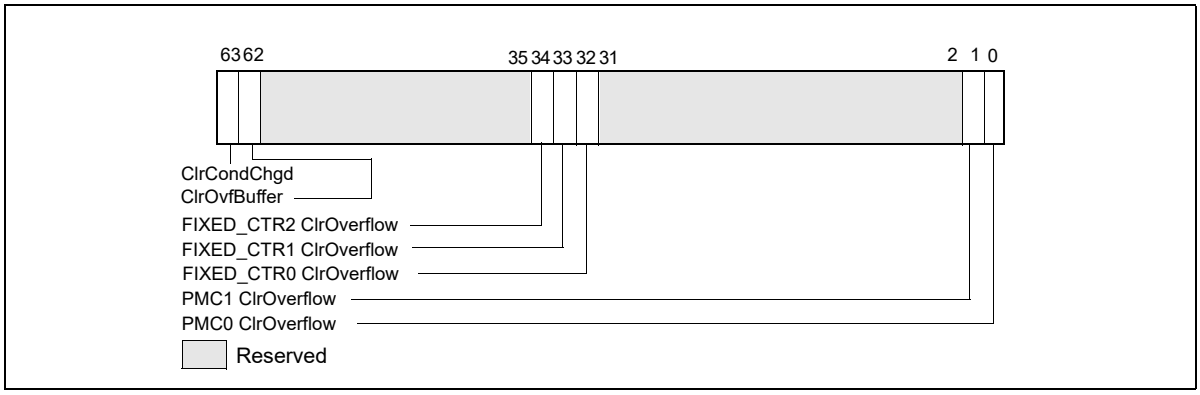


Figure 19-46. Layout of MSR_PERF_GLOBAL_OVF_CTRL MSR

19.6.2.3 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 micro-architecture does not require special MSR programming control (see Section 19.6.3.6, "At-Retirement Counting"), but is limited to IA32_PMC0. See Table 19-83 for a list of events available to processors based on Intel Core micro-architecture.

Table 19-83. 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

19.6.2.4 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 19.6.2.4.2 and Section 17.4.9).

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 19-84. The procedure for detecting availability of PEBS is the same as described in Section 19.6.3.8.1.

Table 19-84. 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

19.6.2.4.1 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 17-8 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 19-84.

19.6.2.4.2 PEBS Record Format

The PEBS record format may be extended across different processor implementations. The IA32_PERF_CAPABILITES 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_CAPABILITES are defined in Table 2-2 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 19.6.3.8).
  - 0001B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS and load latency data. (See Section 19.3.1.1.1).
  - 0010B: PEBS record includes additional information of IA32_PERF_GLOBAL_STATUS, load latency data, and TSX tuning information. (See Section 19.3.6.2).
  - 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 19.3.8.1.1).
  - 0100B: PEBS record contents are defined by elections in MSR_PEBS_DATA_CFG. (See Section 19.9.2.3). The PEBS Configuration Buffer is defined as shown in Figure 19-64 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 19.9.2.3). The PEBS Configuration Buffer is defined as shown in Figure 19-64 with Counter Reset fields allocation for 32 general-purpose counters followed by 16 fixed-function counters.

19.6.2.4.3 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 17.4.9.1, "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 19-85.

Table 19-85. 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 PMC0 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 PMC0 bit in IA32_PEBS_ENABLE MSR (3F1H).	Configure MSR_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT and MSR_PEBS_MATRIX_HORZ as needed.
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.

19.6.2.4.4 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).

19.6.3 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. CCCRs 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 19-86 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 19-86. 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

Table 19-86. 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_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_SAAT_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_SAAT_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
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

Table 19-86. 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_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 μops 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.

19.6.3.1 ESCR MSRs

The 45 ESCR MSRs (see Table 19-86) allow software to select specific events to be countered. Each ESCR is usually associated with a pair of performance counters (see Table 19-86) and each performance counter has several ESCRs associated with it (allowing the events counted to be selected from a variety of events).

Figure 19-47 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.)

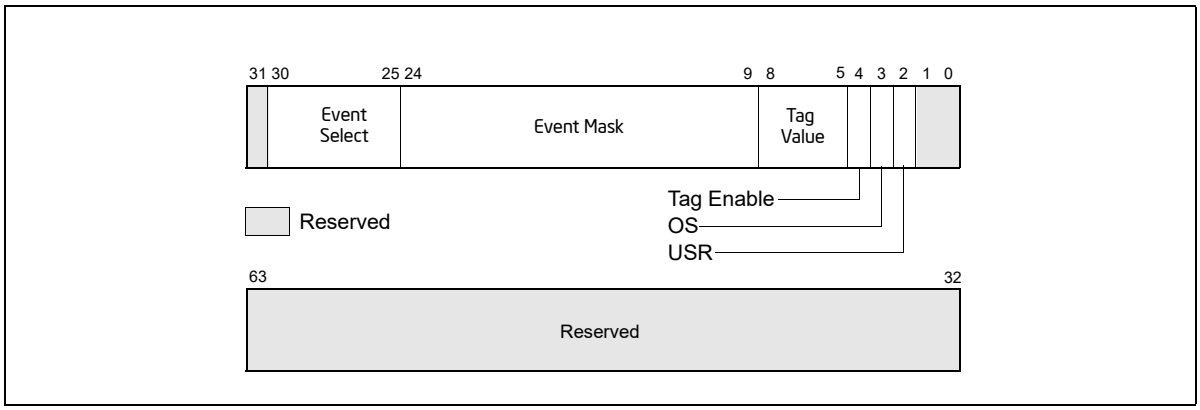


Figure 19-47. Event Selection Control Register (ESCR) for Pentium 4 and Intel® Xeon® Processors without Intel HT Technology Support

- **Tag enable, bit 4 —** When set, enables tagging of μops to assist in at-retirement event counting; when clear, disables tagging. See Section 19.6.3.6, "At-Retirement Counting."

- **Tag value field, bits 5 through 8 —** Selects a tag value to associate with a μop 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 19-86 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.

19.6.3.2 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 19-86). 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 19.6.3.5.6, "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 19-48). 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.

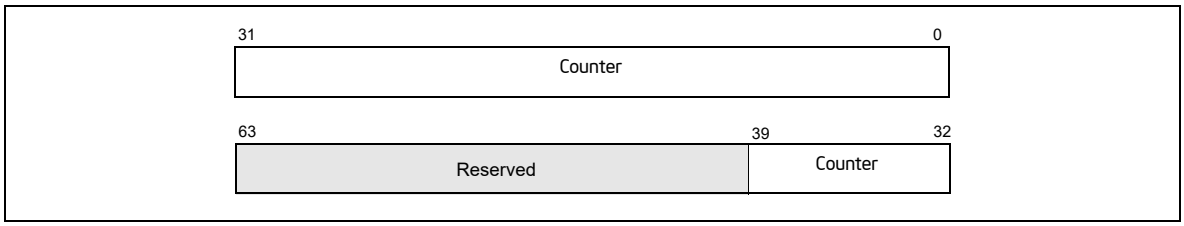


Figure 19-48. Performance Counter (Pentium 4 and Intel® Xeon® Processors)

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.

19.6.3.3 CCCR MSRs

Each of the 18 performance counters has one CCCR MSR associated with it (see Table 19-86). The CCCRs control the filtering and counting of events as well as interrupt generation. Figure 19-49 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 19.6.3.5.2, "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 19.6.3.5.2, "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.

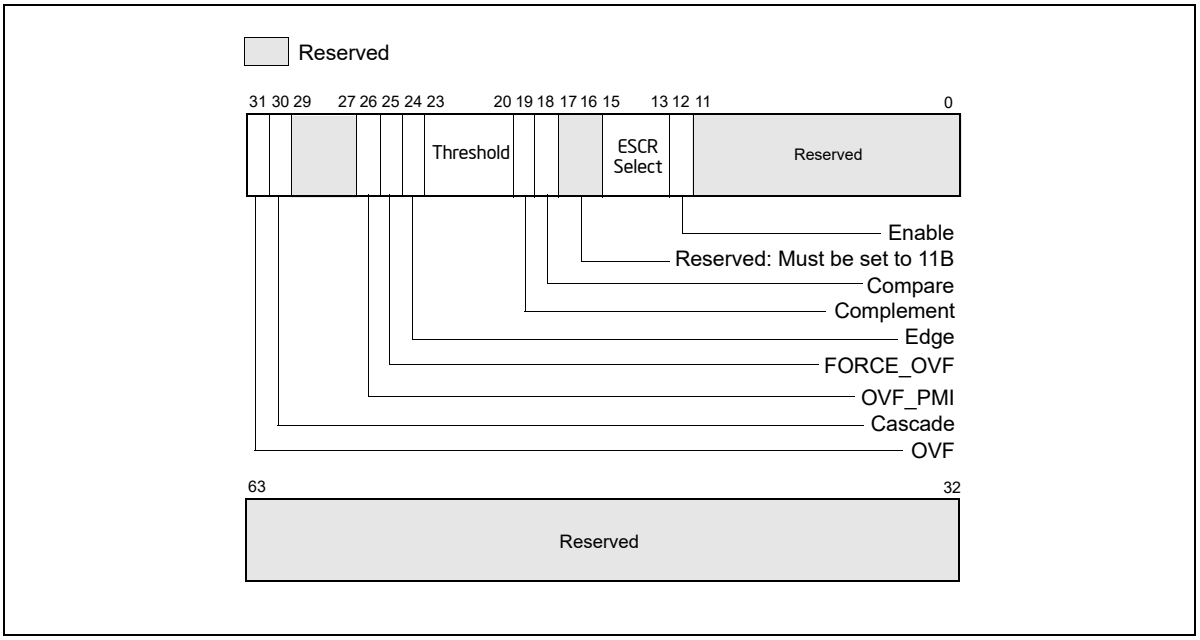


Figure 19-49. Counter Configuration Control Register (CCCR)

- **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 19.6.3.2, "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 19.6.3.5, "Programming the Performance Counters for Non-Retirement Events."

19.6.3.4 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 17.4.5, "Branch Trace Store (BTS)," and Section 19.6.3.8, "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 17.4.9, "BTS and DS Save Area."

19.6.3.5 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 19-86; 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.

19.6.3.5.1 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 19-87 gives an example of one of the events.

Table 19-87. 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]

Table 19-87. Event Example (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	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 19-86 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 19-86.
- **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 19-86.
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 19.6.3.5.2, "Filtering Events."

19.6.3.5.2 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 3 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 counters 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, than 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 19-50 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 19.6.3.5.1, "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 19.6.3.5.3, "Starting Event Counting."

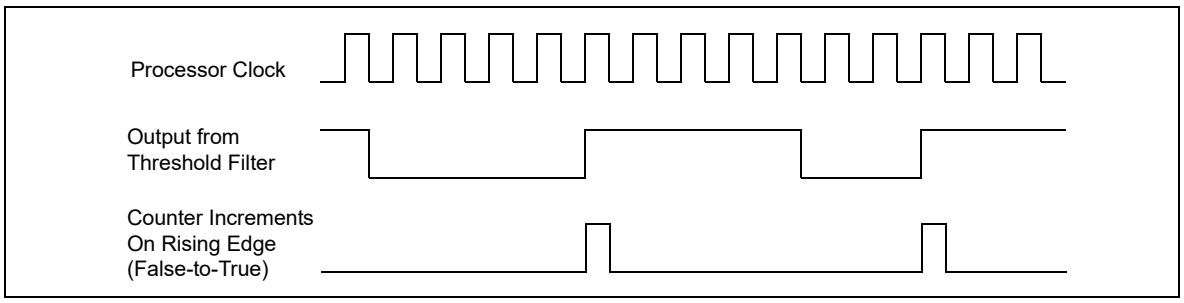


Figure 19-50. Effects of Edge Filtering

19.6.3.5.3 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 19.6.3.5.5, "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 19.6.3.5.2, "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 19.6.3.5.4, "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 19.6.3.5.6, "Cascading Counters").

19.6.3.5.4 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 19.6.3.2, "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 19.6.3.5.3, "Starting Event Counting."

10. To read a performance counters current event count, execute the RDPMC instruction with the counter number obtained from Table 19-86 used as an operand.

This setup procedure is continued in the next section, Section 19.6.3.5.5, "Halting Event Counting."

19.6.3.5.5 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 19.6.3.5.4, "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.

19.6.3.5.6 Cascading Counters

As described in Section 19.6.3.2, "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 19-86). 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 19-49 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 19-1. 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 19.6.3.5.1, "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 19.6.3.5.8, "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.

19.6.3.5.7 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 19-88.

Table 19-88. CCR Names and Bit Positions

CCCR Name:Bit Position	Bit Name	Description
MSR_IQ_CCCR1|2:11	Reserved	
MSR_IQ_CCCR0:11	CASCNT4INTO0	Allow counter 4 to cascade into counter 0
MSR_IQ_CCCR3:11	CASCNT5INTO3	Allow counter 5 to cascade into counter 3
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 19-2. 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 FFFFF000H 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.

19.6.3.5.8 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 19.6.3.5.6, "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.

19.6.3.5.9 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).

19.6.3.6 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 μops 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 μops along the path that results in committed architectural state changes as required by the program being executed. Thus instructions and μops 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 μops that are retired based on the characterization of bogus" versus non-bogus.
- **Tagging —** Tagging is a means of marking μops 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 μop and a direct count of the event would not provide an indication of how many μops encountered that event.

  The tagging mechanisms allow a μop 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 μop, rather than once per event. For example, a μop may encounter a cache miss more than once during its life time, but a "Miss Retired" metric (that counts the number of retired μops that encountered a cache miss) will increment only once for that μop. 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 Pentium 4 Processor Optimization Reference Manual* (see Section 1.4, "Related Literature").
- **Replay —** To maximize performance for the common case, the Intel NetBurst microarchitecture aggressively schedules μops 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, μops must be reissued. The mechanism that the Pentium 4 and Intel Xeon processors use for this reissuing of μops 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 μops before they begin and are costly.

19.6.3.6.1 Using At-Retirement Counting

Processors based on Intel NetBurst microarchitecture allow counting both events and μops that encountered a specified event. For a subset of the at-retirement events, a μop 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 μops tagged with that mechanism:

- **Front-end tagging —** This mechanism pertains to the tagging of μops 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 μops that encountered execution events (for example, instruction types) and are counted with the Execution_Event event.

- **Replay tagging —** This mechanism pertains to tagging of μops 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 μop that has been tagged using one mechanism will not be detected with another mechanism's tagged-μop detector. For example, if μops are tagged using the front-end tagging mechanisms, the Replay_event will not count those as tagged μops unless they are also tagged using the replay tagging mechanism. However, execution tags allow up to four different types of μops 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 μops 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 μop and count tagged μops.

19.6.3.6.2 Tagging Mechanism for Front_end_event

The Front_end_event counts μops that have been tagged as encountering any of the following events:

- **μop decode events —** Tagging μops for μop decode events requires specifying bits in the ESCR associated with the performance-monitoring event, Uop_type.
- **Trace cache events —** Tagging μops 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.

19.6.3.6.3 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 μops 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 μop. 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 μops. 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 19.6.3.8.3, "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.

19.6.3.7 Tagging Mechanism for Replay_event

The replay mechanism enables tagging of μops for a subset of all replays before retirement. Use of the replay mechanism requires selecting the type of μop 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 17.4.9). 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 μops.

19.6.3.8 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 17.4.5, "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 17.4.9, "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_PERFEVTSEL0 MSRs (See Section 19.6.2.4).

19.6.3.8.1 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.

19.6.3.8.2 Setting Up the DS Save Area

Section 17.4.9.2, "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.

19.6.3.8.3 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 17-5) 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.

19.6.3.8.4 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 17.4.9.5, "Writing the DS Interrupt Service Routine," for guidelines for writing the DS ISR.

19.6.3.8.5 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.

19.6.3.9 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.

19.6.4 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 19.6.3. 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.

19.6.4.1 ESCR MSRs

Figure 19-51 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.

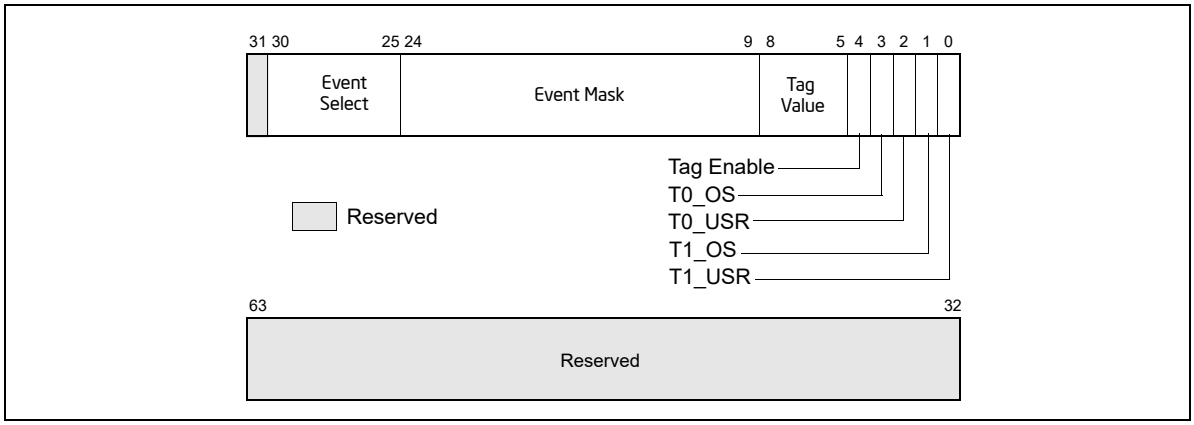


Figure 19-51. Event Selection Control Register (ESCR) for the Pentium 4 Processor, Intel® Xeon® Processor and Intel® Xeon® Processor MP Supporting Hyper-Threading Technology

- **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 μops to assist in at-retirement event counting; when clear, disables tagging. See Section 19.6.3.6, "At-Retirement Counting."
- **Tag value field, bits 5 through 8 —** Selects a tag value to associate with a μop 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 8.4.5, "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 19.6.4.4, "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.

19.6.4.2 CCCR MSRs

Figure 19-52 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.

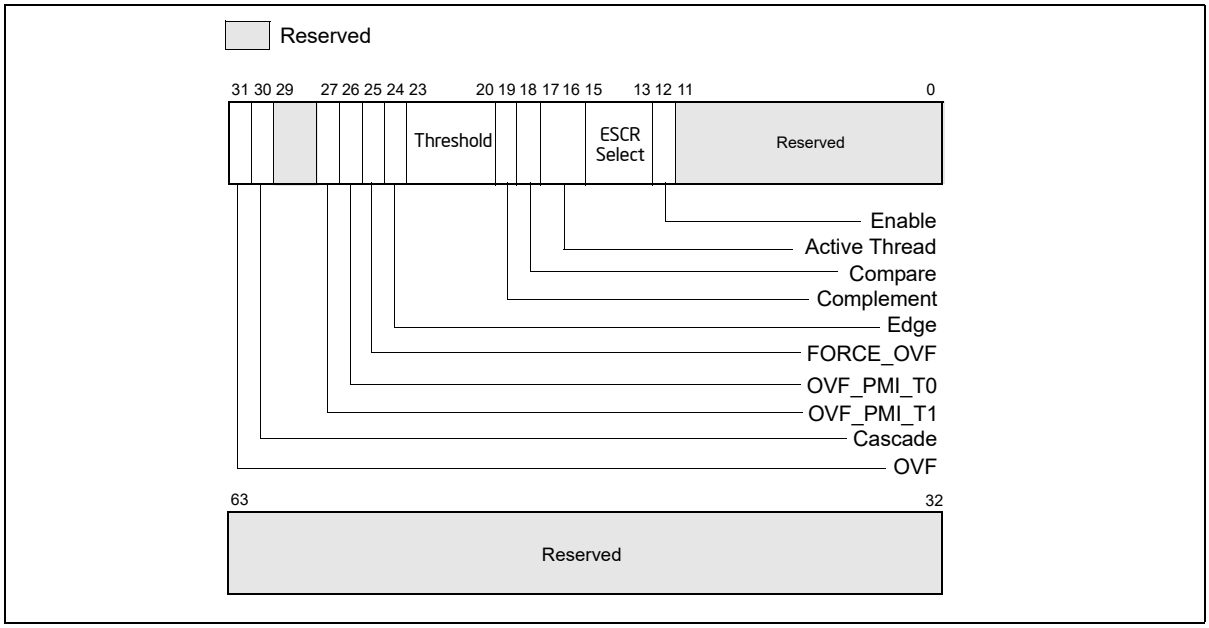


Figure 19-52. Counter Configuration Control Register (CCCR)

- **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 19.6.3.5.2, "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 19.6.3.5.2, "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 19.6.3.2, "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.

19.6.4.3 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.

19.6.4.4 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 19-89) 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 19-89. 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) or 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 19-90.

Table 19-90. 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

19.6.4.5 Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture

19.6.4.5.1 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.

19.6.4.5.2 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.

19.6.5 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 19.6.4, "Performance Monitoring and Intel Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture").

19.6.6 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 CPUID 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 19.1 and Section 19.6.4) 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 19-53.

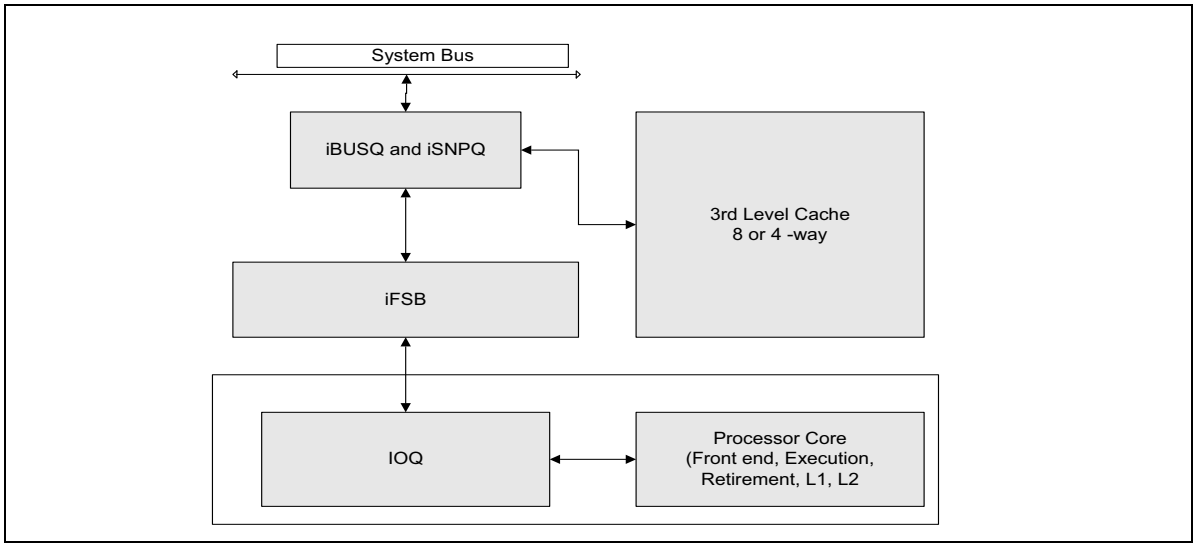


Figure 19-53. Block Diagram of 64-bit Intel® Xeon® Processor MP with 8-MByte L3

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 19-54.

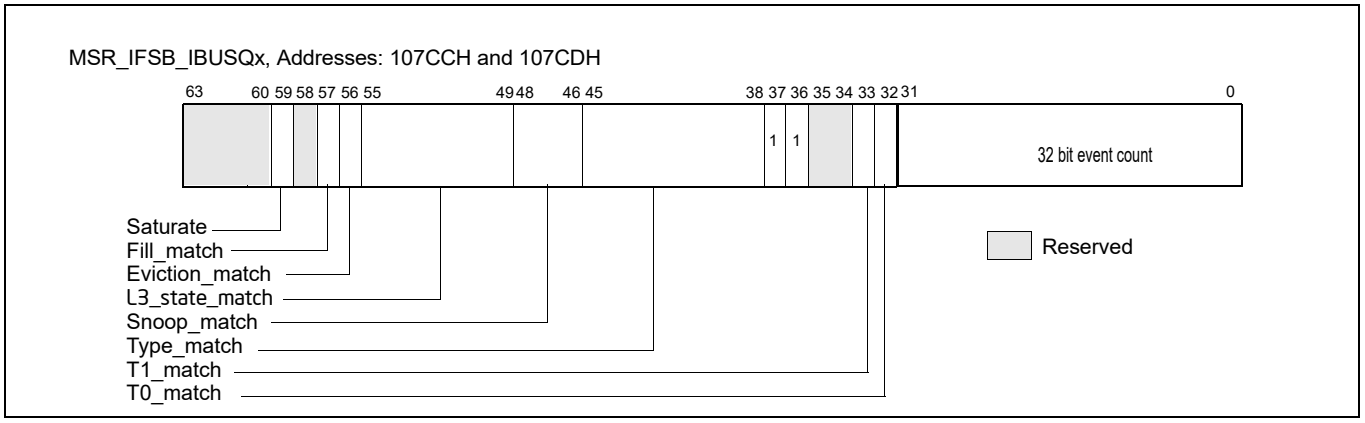


Figure 19-54. MSR_IFSB_IBUSQx, Addresses: 107CCH and 107CDH

- **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 19-55.

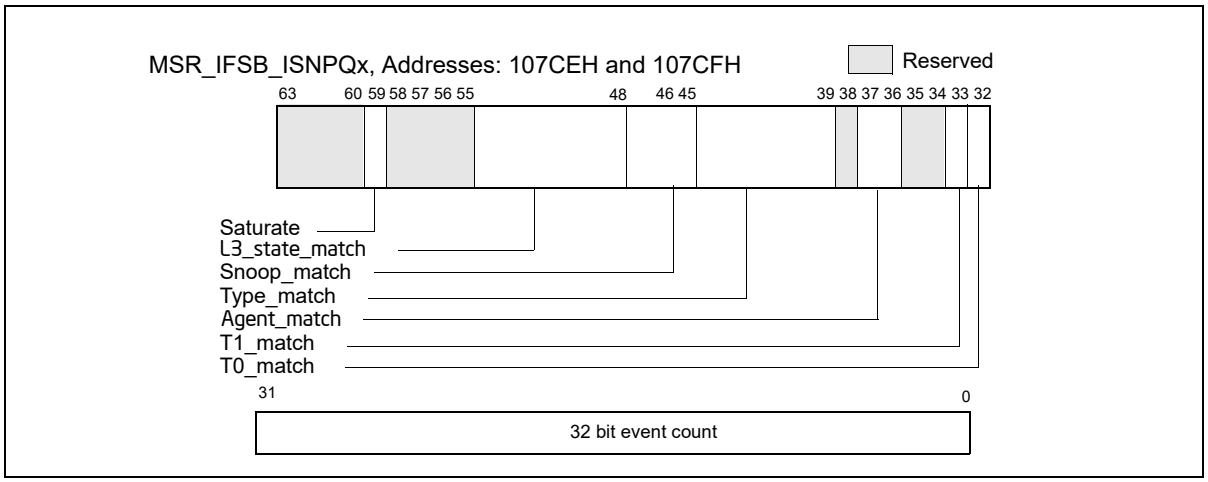


Figure 19-55. MSR_IFSB_ISNPQx, Addresses: 107CEH and 107CFH

- **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 19-56.

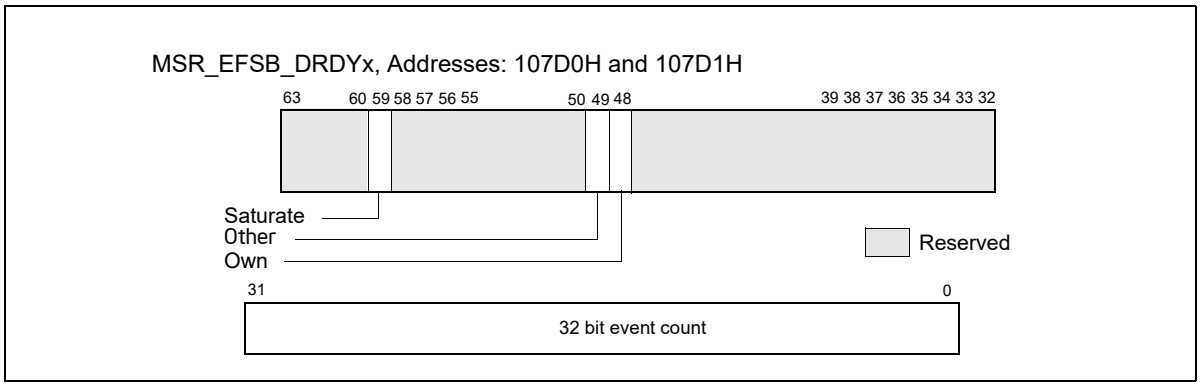


Figure 19-56. MSR_EFSB_DRDYx, Addresses: 107D0H and 107D1H

- **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 19-57.

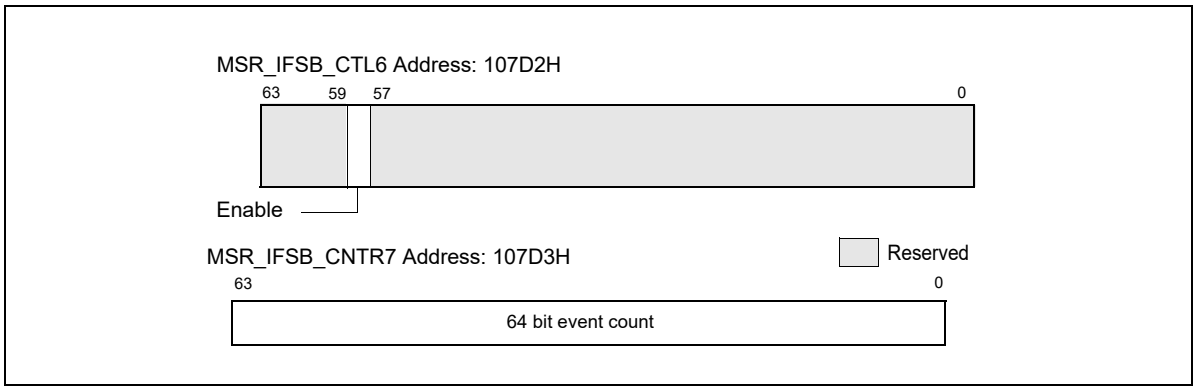


Figure 19-57. MSR_IFSB_CTL6, Address: 107D2H; MSR_IFSB_CNTR7, Address: 107D3H

19.6.7 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 CPUID signature is indicated by DisplayFamily_DisplayModel value of 06_1DH (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*). 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 CPUID 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 19-58 for the block configuration of six processor cores and the L3/Caching bus controller sub-system in Intel Xeon processor 7400 series. Figure 19-58 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.

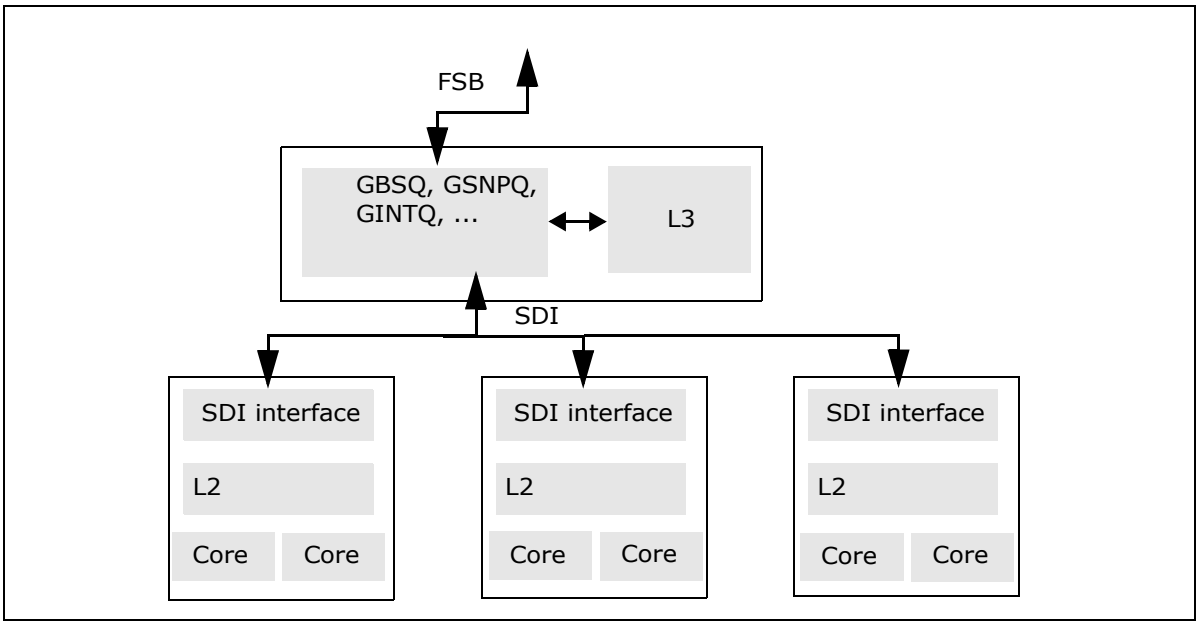


Figure 19-58. Block Diagram of the Intel® Xeon® Processor 7400 Series

Almost all of the performance monitoring capabilities available to processor cores with the same CPUID signatures (see Section 19.1 and Section 19.6.4) 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.

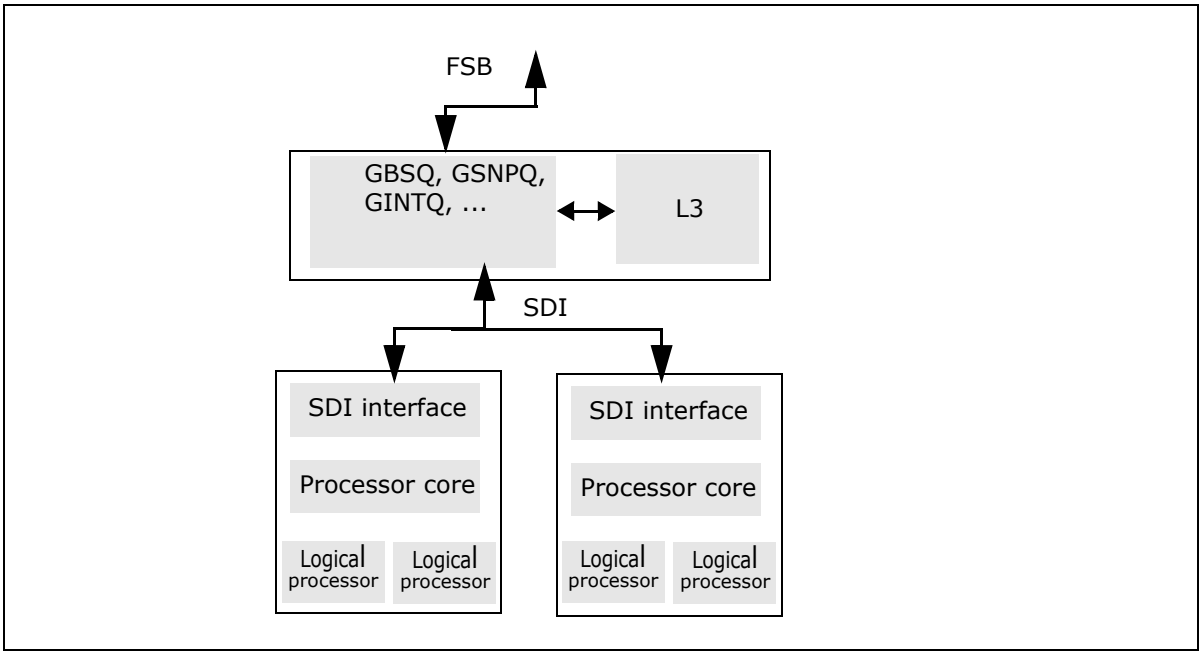


Figure 19-59. Block Diagram of the Intel® Xeon® Processor 7100 Series

19.6.7.1 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.

19.6.7.2 GBSQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL0 and MSR_EMON_L3_CTR_CTL1 is given in Figure 19-60. 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.

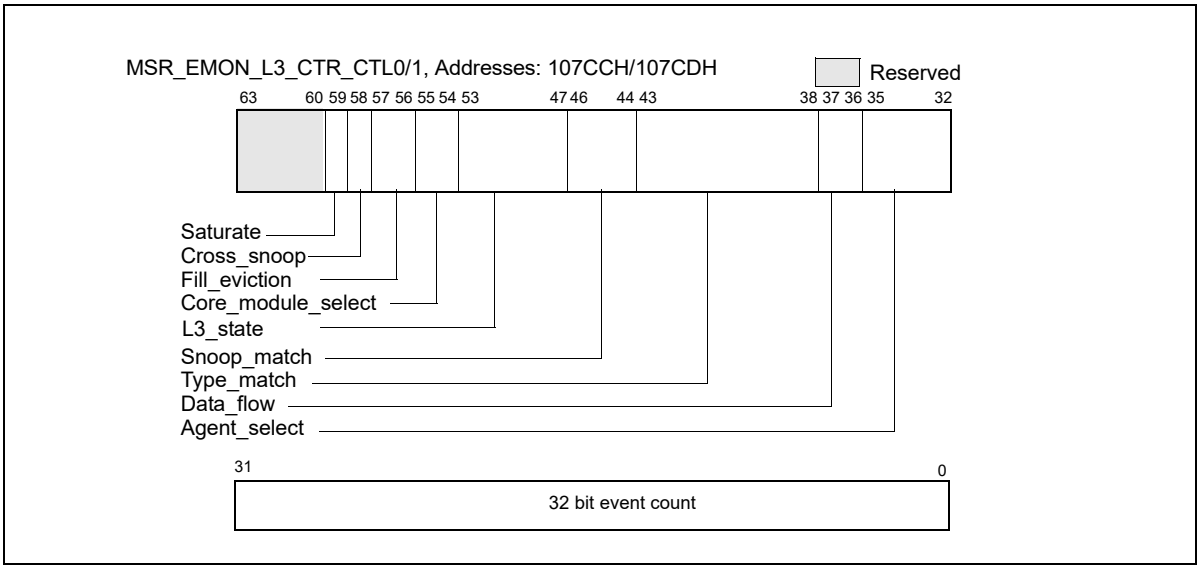


Figure 19-60. MSR_EMON_L3_CTR_CTL0/1, Addresses: 107CCH/107CDH

- 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.

19.6.7.3 GSNPQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL2 and MSR_EMON_L3_CTR_CTL3 is given in Figure 19-61. 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.

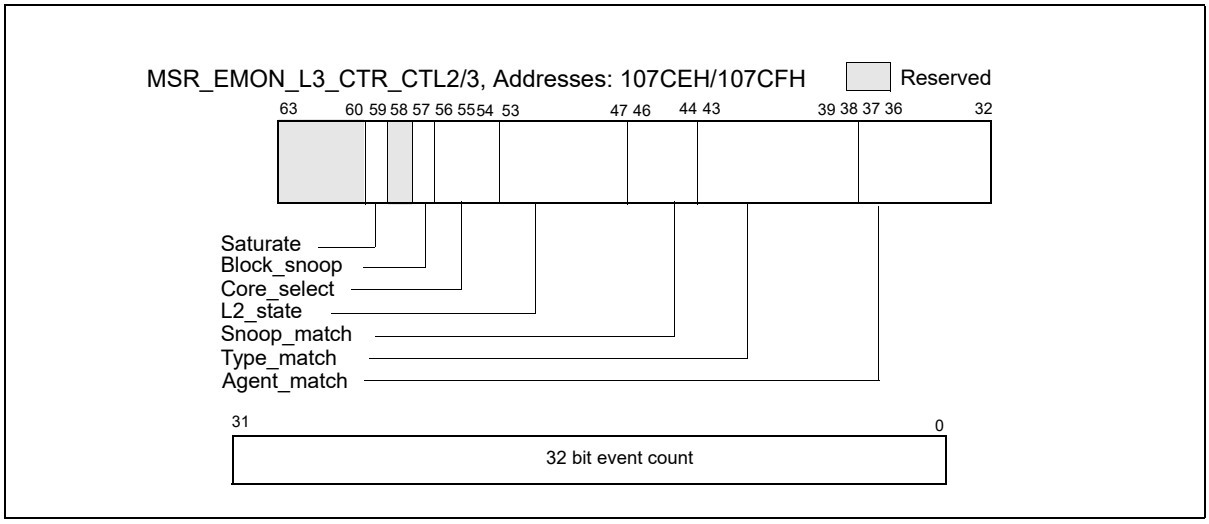


Figure 19-61. MSR_EMON_L3_CTR_CTL2/3, Addresses: 107CEH/107CFH

19.6.7.4 FSB Event Interface

The layout of MSR_EMON_L3_CTR_CTL4 through MSR_EMON_L3_CTR_CTL7 is given in Figure 19-62. 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) 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.

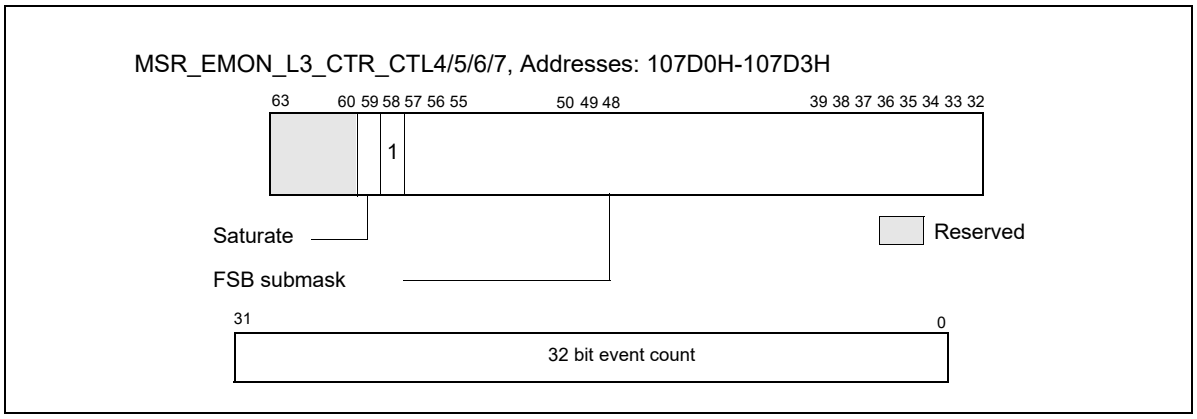


Figure 19-62. MSR_EMON_L3_CTR_CTL4/5/6/7, Addresses: 107D0H-107D3H

19.6.7.4.1 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.

19.6.7.5 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 19-60 for example) is set, the event logic forces the value FFFF_FFFFH into the event count field instead of incrementing it.

19.6.8 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 RDPMC (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.

19.6.8.1 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 19-63 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.

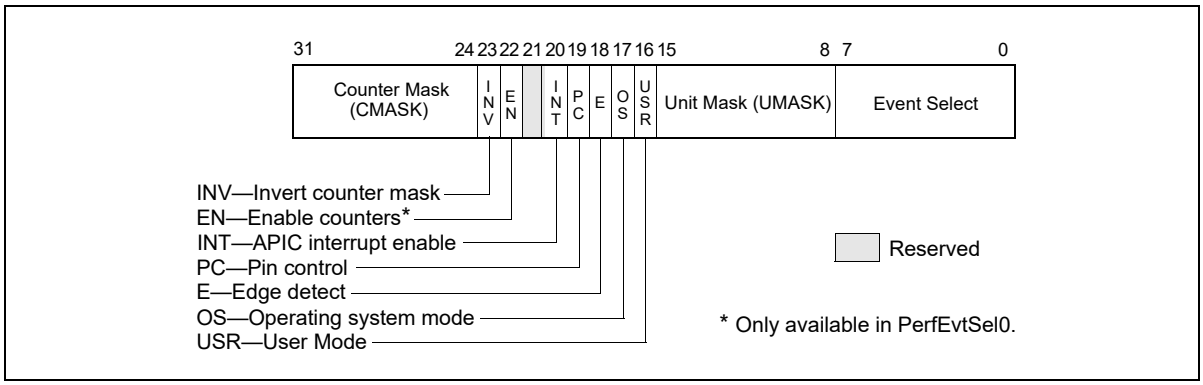


Figure 19-63. PerfEvtSel0 and PerfEvtSel1 MSRs

- **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 PM*i* pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PM*i* 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.

19.6.8.2 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.

19.6.8.3 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.

19.6.8.4 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 19.6.8.3, "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.

19.6.8.5 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.

19.6.9 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.

19.6.9.1 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 19-64). 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.

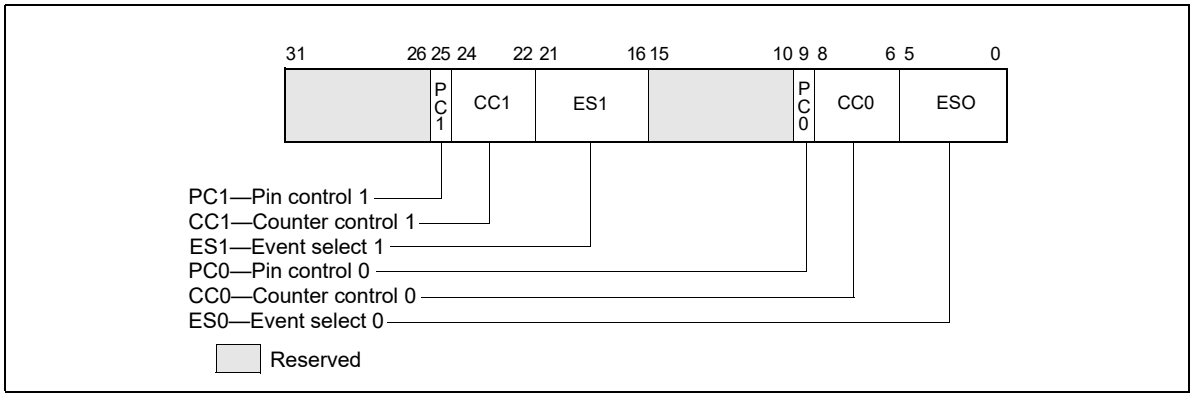


Figure 19-64. CESR MSR (Pentium Processor Only)

- **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.

19.6.9.2 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 then $2^{40} - 1$. 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.

19.6.9.3 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.

19.7 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 19.7.2 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 17.17, "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 17.17, "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.

19.7.1 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.

19.7.2 Cycle Counting and Opportunistic Processor Operation

As a result of the state transitions due to opportunistic processor performance operation (see Chapter 14, "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.

19.7.3 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 (see Table 3-8 "Information Returned by CPUID Instruction"), the nominal TSC frequency can be determined by using the following equation:

Nominal TSC frequency = (CPUID.15H.ECX[31:0] * CPUID.15H.EBX[31:0]) ÷ CPUID.15H.EAX[31:0]

For Intel processors in which CPUID.15H.EBX[31:0] ÷ CPUID.0x15.EAX[31:0] is enumerated but CPUID.15H.ECX is not enumerated, Table 19-91 can be used to look up the nominal core crystal clock frequency.

Table 19-91. Nominal Core Crystal Clock Frequency

Processor Families/Processor Number Series[1]	Nominal Core Crystal Clock Frequency
Intel® Xeon® Processor Scalable 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:

1. 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.

19.7.3.1 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.

19.7.3.2 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.

19.7.3.3 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*.

19.7.3.4 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.

19.8 IA32_PERF_CAPABILITIES MSR ENUMERATION

The layout of IA32_PERF_CAPABILITIES MSR is shown in Figure 19-65; 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 17.4.8.1.
- IA32_PERF_CAPABILITIES.PEBSTrap[6]: Trap/Fault-like indicator of PEBS recording assist; see Section 19.6.2.4.2.
- IA32_PERF_CAPABILITIES.PEBSArchRegs[7]: Indicator of PEBS assist save architectural registers; see Section 19.6.2.4.2.
- IA32_PERF_CAPABILITIES.PEBS_FMT[bits 11:8]: Specifies the encoding of the layout of PEBS records; see Section 19.6.2.4.2.
- IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[12]: Indicates IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if 1, see Section 19.8.1.
- 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 19.2.6.
- 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 19.6.2.4.2.
  - 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 19.9.2.3.
  - 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 19.3.9.3.
- IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16]: If set on parts that enumerate support for Intel PT (CPUID.0x7.0.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 19.5.5.2.

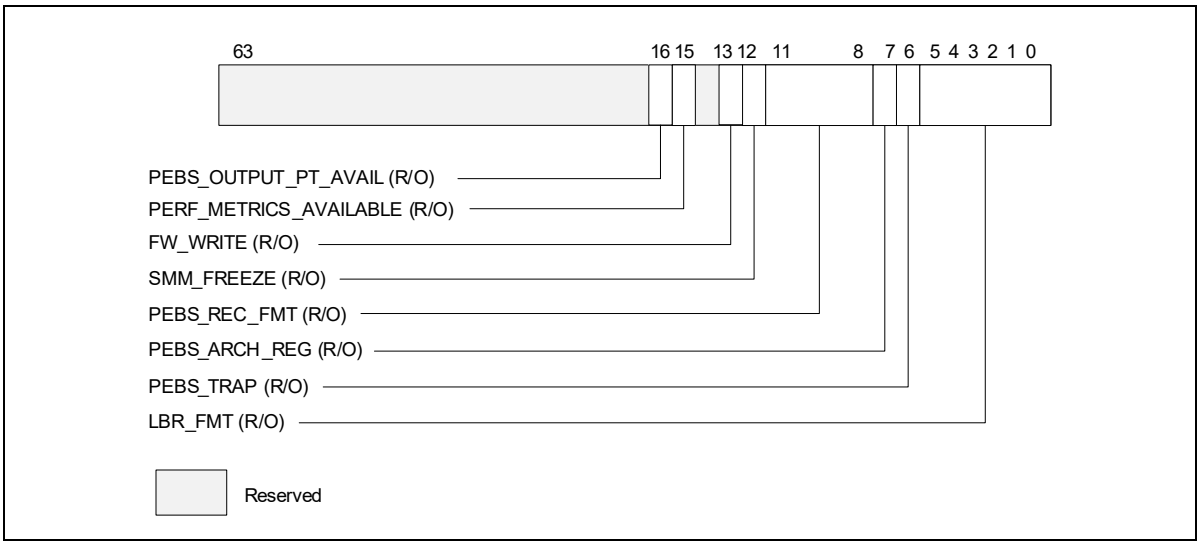


Figure 19-65. Layout of IA32_PERF_CAPABILITIES MSR

19.8.1 Filtering of SMM Handler Overhead

When performance monitoring facilities and/or branch profiling facilities (see Section 17.5, "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.PDCM[bit 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.

19.9 PEBS FACILITY

19.9.1 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).

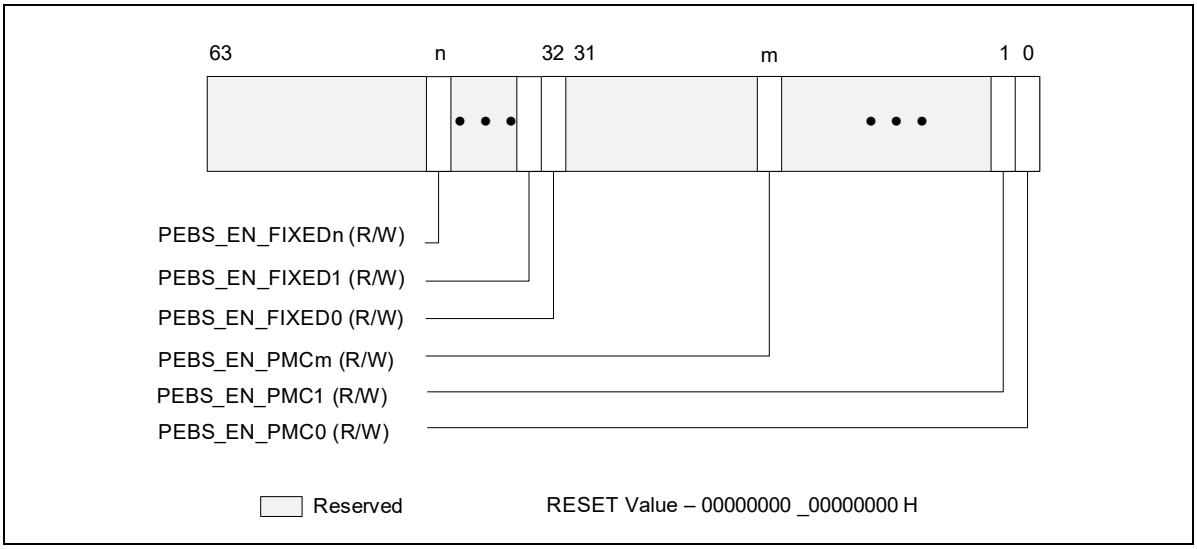


Figure 19-66. Layout of IA32_PEBS_ENABLE MSR

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 19-67. 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).

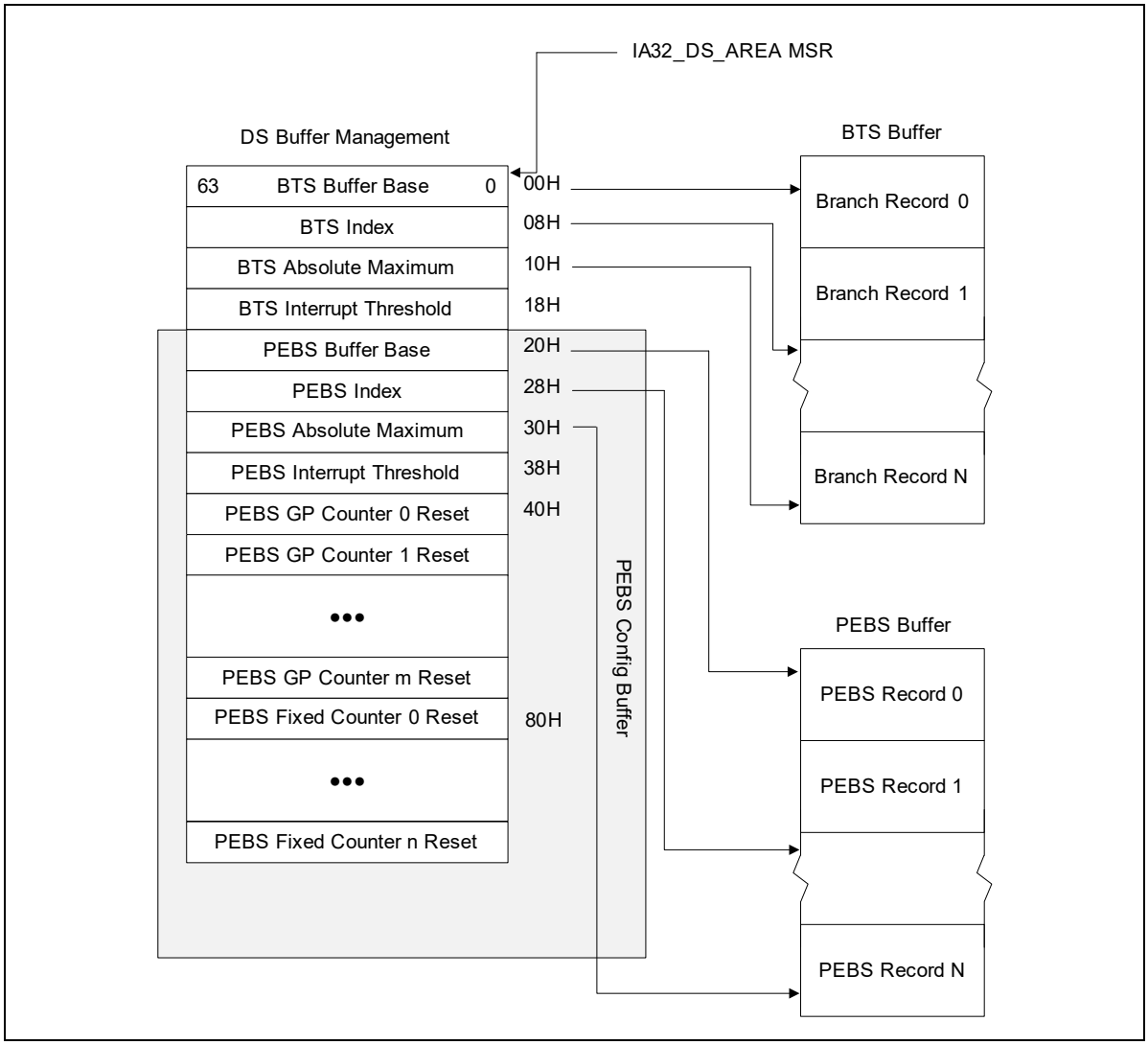


Figure 19-67. PEBS Programming Environment

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.

19.9.2 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)

The PEBS record is restructured where fields are grouped into Basic group, Memory group, GPR group, XMM group and LBR 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.

19.9.2.1 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.

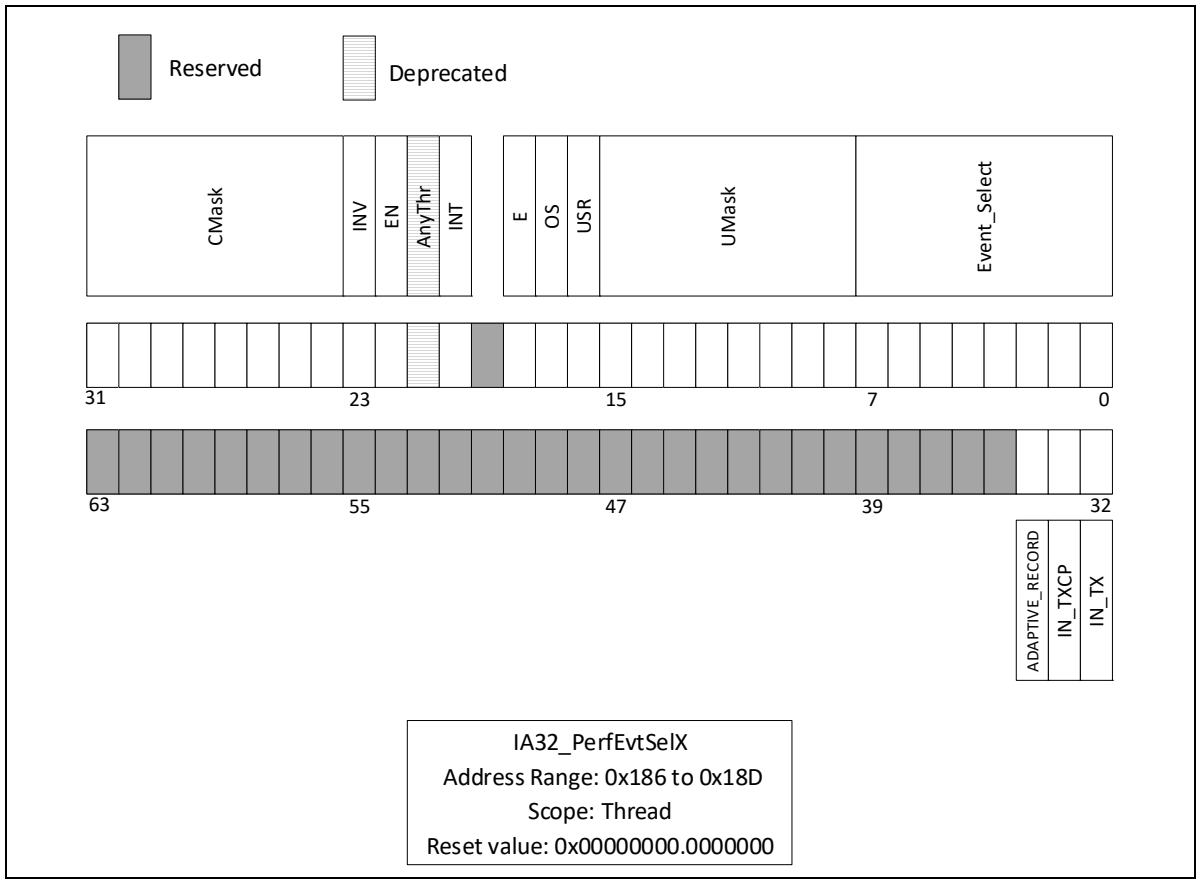


Figure 19-68. Layout of IA32_PerfEvtSelX MSR Supporting Adaptive PEBS

- 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.

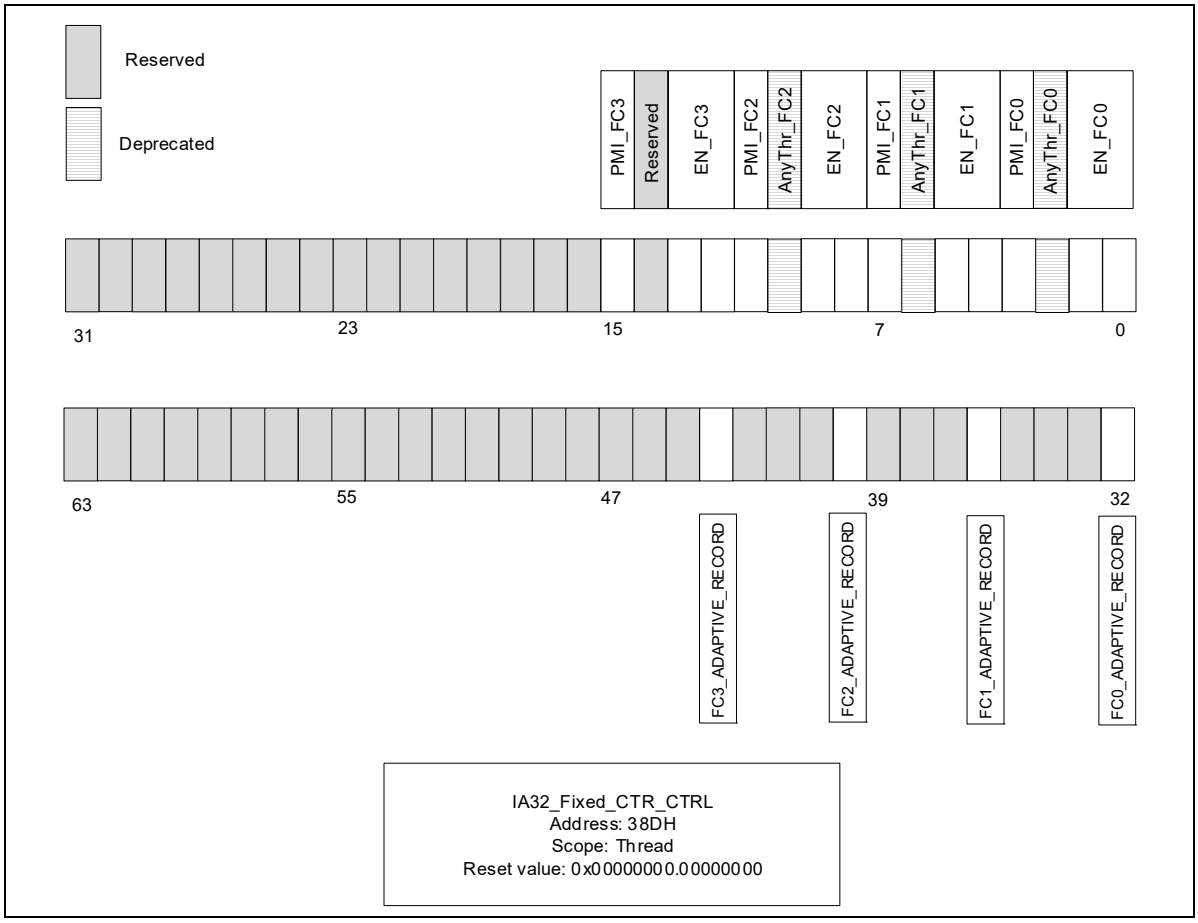


Figure 19-69. Layout of IA32_FIXED_CTR_CTRL MSR Supporting Adaptive PEBS

19.9.2.2 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.

19.9.2.2.1 Basic Info

The Basic group contains essential information for software to parse a record along with several critical fields. It is always collected.

Table 19-92. 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 19-92. 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.

19.9.2.2.2 Memory Access Info

This group contains the legacy PEBS memory-related fields; see Section 19.3.1.1.2.

Table 19-93. 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 MEM_TRANS_RETIRED.* 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 19-4 and Table 19-13. 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 19-14.
Memory Access Latency	[63:0]	When MEM_TRANS_RETIRED.* 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 19.3.6.5.1.

Beginning with 12th generation Intel Core processors, the memory access information group has been updated. New fields added are shaded gray in Table 19-94.

Table 19-94. 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 19-4. 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).

Table 19-94. Updated Memory Access Info Group (Contd.)

Field Name	Sub-field Name	Bits	Description
	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 19.3.6.5.1.

To determine which fields are supported for certain performance monitoring events, consult the Memory Info attribute in the event lists at https://download.01.org/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.

19.9.2.2.3 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 19-95. GPRs in Ice Lake Microarchitecture

Field Name	Bit Width
RFLAGS	[63:0]
RIP	[63:0]
RAX	[63:0]
RCX*	[63:0]
RDX*	[63:0]

Table 19-95. GPRs in Ice Lake Microarchitecture (Contd.)

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.

19.9.2.2.4 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 19-96. XMMs

Field Name	Bit Width
XMM0	[127:0]
…	…
XMM15	[127:0]

19.9.2.2.5 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 19-97. 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 17.12.1, “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.

19.9.2.3 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.

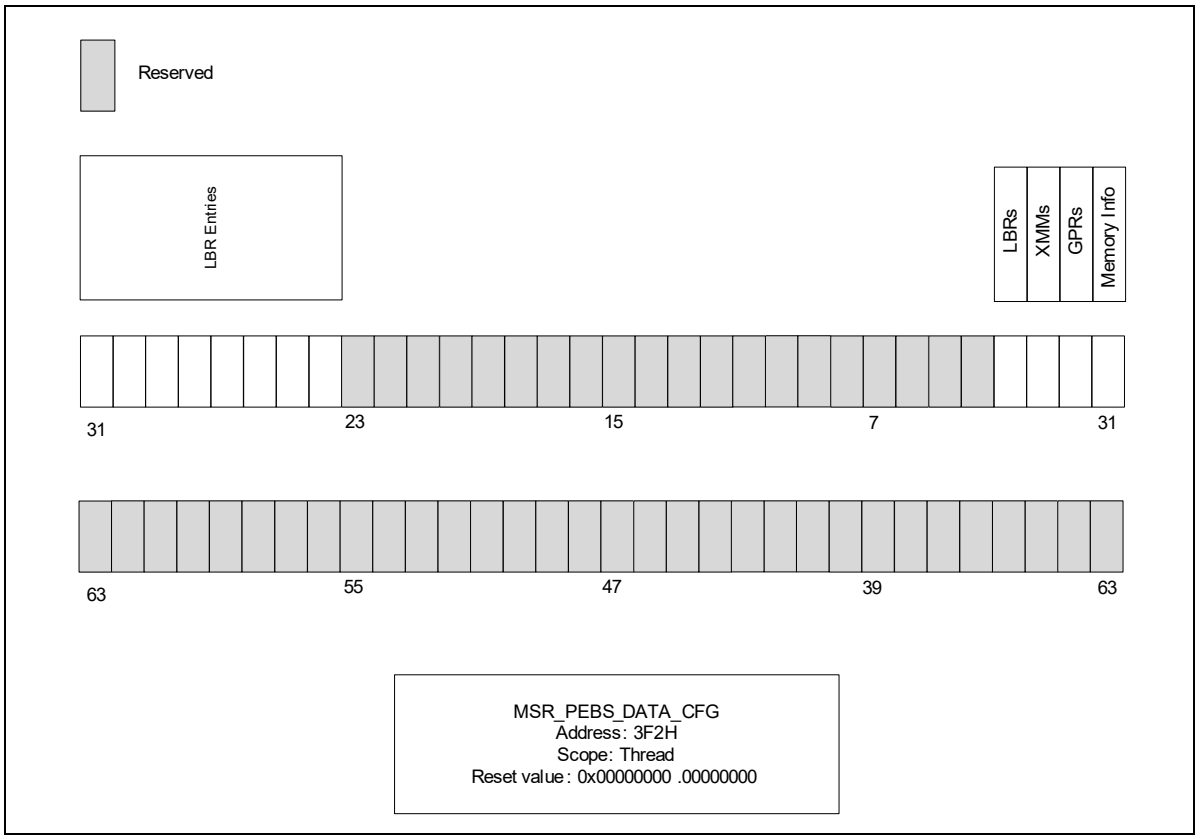


Figure 19-70. MSR_PEBS_DATA_CFG

Table 19-98. MSR_PEBS_CFG Programming[1]

Bit	Bit Index	Access	Description
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.
GPRs	1	R/W	Setting this bit will capture the contents of the General Purpose registers in the PEBS record.
XMMs	2	R/W	Setting this bit will capture the contents of the XMM registers in the PEBS record.
LBRs	3	R/W	Setting this bit will capture LBR TO, FROM and INFO in the PEBS record.
Reserved[2]	23:4	NA	Reserved

Table 19-98. MSR_PEBS_CFG Programming[1]

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).

NOTES:

1. A write to the MSR will be ignored when IA32_MISC_ENABLE.PERFMON_AVAILABLE is zero (default).
2. Writing to the reserved bits will cause a GP fault.

19.9.2.4 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 19-99. 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
0x40	GPRs	RFLAGS	
0x48		RIP	
0x50		RAX	
...		...	
0x88		RDI	
0x90		R8	
...		...	
0xC8		R15	
0xD0	XMMs	XMM0	New
...		...	
0x1C0		XMM15	

Table 19-99. PEBS Record Example 1

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 19-100. 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	

19.9.3 Precise Distribution of Instructions Retired (PDIR) Facility

Precise Distribution of Instructions Retired Facility is available via PEBS on some microarchitectures. Refer to Section 19.3.4.4.4. Counters that support PDIR also vary. See the processor specific sections for availability.

19.9.4 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 19.3.4.4.4

In the Goldmont microarchitecture, the mechanism applies to all precise events including, INST_RETIRED, except for UOPS_RETIRED. 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_RETIRED, 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).

19.9.5 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.

19.9.6 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 events[1]. The legacy PEBS behavior applies to counters that do not support PDist, unless specified otherwise. PDist requires that the INV, ANY, E, 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 lesser than 127 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 whether PEBS is enabled for that counter via IA32_PEBS_ENABLE) or counter value (MSR write to IA32_PMCx and IA32_A_PMCx).

1. To determine whether an event is precise or supports PDist, consult the relevant attribute in the event lists at https://download.01.org/perfmon/.

19.9.7 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 19-94, "*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 19-101. In the descriptions, local memory refers to system memory physically attached to a 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 19-94.

Table 19-101. Data Source Encoding for Memory Accesses (Ice Lake and Later Microarchitectures)

Encoding	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.

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 19.3.4.4.2 for further implementation details of Load Latency.

19.9.8 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 flagged[1].

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 19-48.

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*.

20. Updates to Chapter 24, Volume 3C

Change bars and green text show changes to Chapter 24 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

Changes to this chapter: Architectural LBR and HLAT additions throughout chapter. Update to Section 24.6.2, "Processor-Based VM-Execution Controls". Update to Section 24.7.1, "VM-Exit Controls".

CHAPTER 24
VIRTUAL MACHINE CONTROL STRUCTURES

24.1 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 24.4.2) 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".
- 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).

---

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. 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.
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.

Figure 24-1 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 24.11.3.

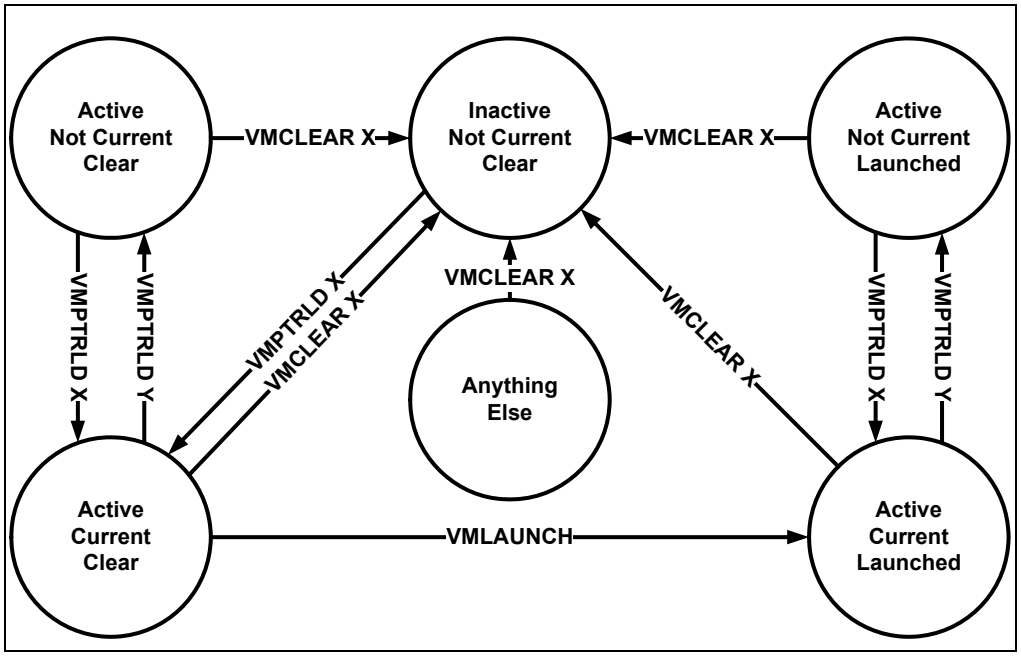


Figure 24-1. States of VMCS X

Because a shadow VMCS (see Section 24.10) cannot be used for VM entry, the launch state of a shadow VMCS is not meaningful. Figure 24-1 does not illustrate all the ways in which a shadow VMCS may be made active.

24.2 FORMAT OF THE VMCS REGION

A VMCS region comprises up to 4-KBytes.[1] The format of a VMCS region is given in Table 24-1.

Table 24-1. Format of the VMCS Region

Byte Offset	Contents
0	Bits 30:0: VMCS revision identifier Bit 31: shadow-VMCS indicator (see Section 24.10)
4	VMX-abort indicator
8	VMCS data (implementation-specific format)

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).

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 24.10).

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 24.6.2) 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 24.10). 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_CTLS2 (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 27.7). 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 24.3 through Section 24.9. To ensure proper behavior in VMX operation, software should maintain the VMCS region and related structures (enumerated in Section 24.11.4) in writeback cacheable memory. Future implementations may allow or require a different memory type[3]. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

24.3 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]

The VM-execution control fields, the VM-exit control fields, and the VM-entry control fields are sometimes referred to collectively as VMX controls.

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).

24.4 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 26.3.2 and Section 27.3 for details.

24.4.1 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 24-2 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 24-2. 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
11:8	Reserved
12	AVL — Available for use by system software

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 6.14, "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 10-1 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Table 24-2. Format of Access Rights (Contd.)

Bit Position(s)	Field
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 26.3.1.2).

- 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.
  - 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.

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_PKRS (64 bits). This field is supported only on processors that support the 1-setting of the "load PKRS" VM-entry control.
- 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.

24.4.2 Guest Non-Register State

In addition to the register state described in Section 24.4.1, 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 24-3.

Table 24-3. 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 6.8.3, "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 31.2, "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.

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 27.1.
2. A triple fault occurs when a logical processor encounters an exception while attempting to deliver a double fault.

Table 24-3. Format of Interruptibility State (Contd.)

Bit Position(s)	Bit Name	Notes
3	Blocking by NMI	See Section 6.7.1, "Handling Multiple NMIs," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* and Section 31.8, "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 25.3 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 24.6.1) 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 that is incident to delivery of an event injected by VM entry leaves this bit unmodified.
31:5	Reserved	VM entry will fail if these bits are not 0. See Section 26.3.1.5.

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.

- **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.[1] This field contains information about such exceptions. This field is described in Table 24-4.

Table 24-4. 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.
11:4	Reserved	VM entry fails if these bits are not 0. See Section 26.3.1.5.
12	Enabled breakpoint	When set, this bit indicates that at least one data or I/O breakpoint was met and was enabled in DR7.
13	Reserved	VM entry fails if this bit is not 0. See Section 26.3.1.5.
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 26.3.1.5.

1. For example, execution of a MOV to SS or a POP to SS may inhibit some debug exceptions for one instruction. See Section 6.8.3 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 6-2 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Table 24-4. Format of Pending-Debug-Exceptions (Contd.)

Bit Position(s)	Bit Name	Notes
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 16.3.7, "RTM-Enabled Debugger Support," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*).[1]
63:17	Reserved	VM entry fails if these bits are not 0. See Section 26.3.1.5. 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 16 to indicate an RTM-related exception, while this field **sets** the bit to indicate that condition.

- **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 24.10). Otherwise, software should set this field to FFFFFFFF_FFFFFFFFH to avoid VM-entry failures (see Section 26.3.1.5).
- **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 25.5.1 and Section 26.7.4.
- **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 4.4 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 29 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 28.3.6.

24.5 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 27.5).

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 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 27.5 for details of how state is loaded on VM exits.

24.6 VM-EXECUTION CONTROL FIELDS

The VM-execution control fields govern VMX non-root operation. These are described in Section 24.6.1 through Section 24.6.8.

24.6.1 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 24-5 lists the controls. See Chapter 27 for how these controls affect processor behavior in VMX non-root operation.

1. Some asynchronous events cause VM exits regardless of the settings of the pin-based VM-execution controls (see Section 25.2).

Table 24-5. 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 through the guest interrupt-descriptor table (IDT). 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 descriptor 2 of the IDT. This control also determines interactions between IRET and blocking by NMI (see Section 25.3).
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 24-3). This control also interacts with the "NMI-window exiting" VM-execution control (see Section 24.6.2).
6	Activate VMX-preemption timer	If this control is 1, the VMX-preemption timer counts down in VMX non-root operation; see Section 25.5.1. A VM exit occurs when the timer counts down to zero; see Section 25.2.
7	Process posted interrupts	If this control is 1, the processor treats interrupts with the posted-interrupt notification vector (see Section 24.6.8) specially, updating the virtual-APIC page with posted-interrupt requests (see Section 29.6).

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 26.2.1.1).

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.

24.6.2 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.[1] 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).

Table 24-6 lists the primary processor-based VM-execution controls. See Chapter 25 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-6. 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 24.4.2).
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 24.6.5 and Section 25.3).
7	HLT exiting	This control determines whether executions of HLT cause VM exits.
9	INVLPG exiting	This determines whether executions of INVLPG 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.

1. Some instructions cause VM exits regardless of the settings of the processor-based VM-execution controls (see Section 25.1.2), as do task switches (see Section 25.2).

Table 24-6. Definitions of Primary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
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 24.6.7), this control determines whether executions of MOV to CR3 cause VM exits. See Section 25.1.3. 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 29.
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 24.4.2).
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 24.6.4 and Section 25.1.3). 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 25.5.2.
28	Use MSR bitmaps	This control determines whether MSR bitmaps are used to control execution of the RDMSR and WRMSR instructions (see Section 24.6.9 and Section 25.1.3). 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 26.2.1.1).

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 24-7 lists the secondary processor-based VM-execution controls. See Chapter 25 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-7. 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 29.4.
1	Enable EPT	If this control is 1, extended page tables (EPT) are enabled. See Section 28.3.
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 29.5.
5	Enable VPID	If this control is 1, cached translations of linear addresses are associated with a virtual-processor identifier (VPID). See Section 28.1.
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 29.4 and Section 29.5.
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 24.6.13 and Section 25.1.3).
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 25.5.6.
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 24.10 and Section 30.3.
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 24.6.16 and Section 25.1.3.
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 28.3.6.
18	EPT-violation #VE	If this control is 1, EPT violations may cause virtualization exceptions (#VE) instead of VM exits. See Section 25.5.7.
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 32).
20	Enable XSAVES/XRSTORS	If this control is 0, any execution of XSAVES or XRSTORS causes a #UD.
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 28.
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 28.3.4.

Table 24-7. Definitions of Secondary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
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 25.5.4.
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 24.6.5 and Section 25.3).
26	Enable user wait and pause	If this control is 0, any execution of TPAUSE, UMONITOR, or UMWAIT causes a #UD.
28	Enable ENCLV exiting	If this control is 1, executions of ENCLV consult the ENCLV-exiting bitmap to determine whether the instruction causes a VM exit. See Section 24.6.17 and Section 25.1.3.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROCBASED_CTLS2 (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 26.2.1.1).

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 24-8 lists the tertiary processor-based VM-execution controls. See Chapter 25 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-8. 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 4.5.1.
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 28.3.3.2.
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 28.3.3.2.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROCBASED_CTLS3 (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 26.2.1.1).

24.6.3 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 through the IDT, using the descriptor corresponding to 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 25.2 for details.

24.6.4 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 25.1.3 for details. If the bitmaps are used, their addresses must be 4-KByte aligned.

24.6.5 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 25 for a detailed treatment of the behavior of RDTSC, RDTSCP, and RDMSR in VMX non-root operation.

24.6.6 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 27 for details regarding how these fields affect VMX non-root operation.

24.6.7 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 26.2) 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.

24.6.8 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 10.4.4, "Local APIC Status and Location" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* and *Intel® 64 Architecture Processor Topology Enumeration*).[1]

- If the local APIC is in x2APIC mode, it can accesses the local APIC's registers using the RDMSR and WRMSR instructions (see *Intel® 64 Architecture Processor Topology Enumeration*).
- In 64-bit mode, it can access the local APIC's task-priority register (TPR) using the MOV CR8 instruction.

There are five processor-based VM-execution controls (see Section 24.6.2) that control such accesses. There are "use TPR shadow", "virtualize APIC accesses", "virtualize x2APIC mode", "virtual-interrupt delivery", and "APIC-register 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 29.4.

  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 29.

  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 29.3).

  — Accesses to the APIC-access page if, in addition, the "virtualize APIC accesses" VM-execution control is 1 (see Section 29.4).

  — 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 29.5).

  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 29.1.1) 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 29.1.2.

  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 29.1.4 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 29.6 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 29.6 for more information on the use of this field.

1. If the local APIC does not support x2APIC mode, it is always in xAPIC mode.

24.6.9 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 toC0001FFFH. 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 toC0001FFFH. 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 25.1.3 for details. If the bitmaps are used, their address must be 4-KByte aligned.

24.6.10 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 31.15.2. VM entries that return from SMM use this field as described in Section 31.15.4.

24.6.11 Extended-Page-Table Pointer (EPTP)

The **extended-page-table pointer** (EPTP) contains the address of the base of EPT PML4 table (see Section 28.3.2), as well as other EPT configuration information. The format of this field is shown in Table 24-9.

Table 24-9. Format of Extended-Page-Table Pointer

Bit Position(s)	Field
2:0	EPT paging-structure memory type (see Section 28.3.7): 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 28.3.2)
6	Setting this control to 1 enables accessed and dirty flags for EPT (see Section 28.3.5)[2]
7	Setting this control to 1 enables enforcement of access rights for supervisor shadow-stack pages (see Section 28.3.3.2)[3]
11:8	Reserved
N–1:12	Bits N–1:12 of the physical address of the 4-KByte aligned EPT PML4 table[4]
63:N	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. N is the physical-address width supported by the logical processor. 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.

The EPTP exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.

24.6.12 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 28.1 for details regarding the use of this field.

24.6.13 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 25.1.3 for more details regarding PAUSE-loop exiting.

24.6.14 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 24-10 lists the VM-function controls. See Section 25.5.6 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-10. 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 25.5.6.3.

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 26.2.1.1).

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 25.5.6.3).

24.6.15 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 24.10 and Section 30.3).

24.6.16 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 25.1.3 for more information.

24.6.17 ENCLV-Exiting Bitmap

The **ENCLV-exiting bitmap** is a 64-bit field. If the "enable ENCLV exiting" VM-execution control is 1, execution of ENCLV 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 25.1.3 for more information.

24.6.18 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 28.3.6.

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.

24.6.19 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 25.5.7.2.
- **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 25.5.6.3).

24.6.20 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 25.1.3 and Section 25.3).

24.6.21 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 28.3.4). 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 28.3.4.2). The format of this field is shown in Table 24-9.

Table 24-11. Format of Sub-Page-Permission-Table Pointer

Bit Position(s)	Field
11:0	Reserved
N–1:12	Bits N–1:12 of the physical address of the 4-KByte aligned root SPP table
63:N[1]	Reserved

NOTES:

1. N is the processor's physical-address width. Software can determine this width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

The SPPTP exists only on processors that support the 1-setting of the "sub-page write permissions for EPT" VM-execution control.

24.6.22 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 4.5). The format of this field is shown in Table 24-12.

Table 24-12. 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
N–1:12	Guest-physical address (4KB-aligned) of the first HLAT paging structure during linear-address translation.[1]
63:N	Reserved

NOTES:

1. N is the physical-address width supported by the logical processor. 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.

- The HLAT prefix size. The value of this field determines which linear address are subject to HLAT paging. See Section 4.5.1.

These fields exist only on processors that support the 1-setting of the "enable HLAT" VM-execution control.

24.7 VM-EXIT CONTROL FIELDS

The VM-exit control fields govern the behavior of VM exits. They are discussed in Section 24.7.1 and Section 24.7.2.

24.7.1 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 24-13 lists the primary VM-exit controls. See Chapter 27 for complete details of how these controls affect VM exits.

Table 24-13. Definitions of 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.
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 VM-exit interruption-information field, which is marked valid. ▪ If such a VM exit occurs and this control is 0, the interrupt is not acknowledged and the VM-exit interruption-information 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 32).
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.
28	Load CET state	This control determines whether CET-related MSRs and SPP are loaded on VM exit.
29	Load PKRS	This control determines whether the IA32_PKRS MSR is loaded 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 26.2.1.2).

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 entry and VMX non-root operation 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.

Currently, no secondary VM-exit controls are defined, and all bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_EXIT_CTLS2 (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 26.2.1.2).

24.7.2 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 24-14. If the VM-exit MSR-store count is not zero, the address must be 16-byte aligned.

Table 24-14. Format of an MSR Entry

Bit Position(s)	Contents
31:0	MSR index
63:32	Reserved
127:64	MSR data

See Section 27.4 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 24-14). If the VM-exit MSR-load count is not zero, the address must be 16-byte aligned.

See Section 27.6 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).

24.8 VM-ENTRY CONTROL FIELDS

The VM-entry control fields govern the behavior of VM entries. They are discussed in Sections 24.8.1 through 24.8.3.

24.8.1 VM-Entry Controls

The **VM-entry controls** constitute a 32-bit vector that governs the basic operation of VM entries. Table 24-15 lists the controls supported. See Chapter 24 for how these controls affect VM entries.

Table 24-15. 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 31.15.7). 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 32).
18	Load IA32_RTIT_CTL	This control determines whether the IA32_RTIT_CTL MSR is loaded on VM entry.
20	Load CET state	This control determines whether CET-related MSRs and SPP 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.

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 27.2).

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 26.2.1.3).

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.

24.8.2 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 24-14. If the VM-entry MSR-load count is not zero, the address must be 16-byte aligned.

See Section 26.4 for details of how this area is used on VM entries.

24.8.3 VM-Entry Controls for Event Injection

VM entry can be configured to conclude by delivering an event through the IDT (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:

- **VM-entry interruption-information field** (32 bits). This field provides details about the event to be injected. Table 24-16 describes the field.

Table 24-16. Format of the VM-Entry Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Reserved 2: Non-maskable interrupt (NMI) 3: Hardware exception (e.g,. #PF) 4: Software interrupt (INT *n*) 5: Privileged software exception (INT1) 6: Software exception (INT3 or INTO) 7: Other event
11	Deliver error code (0 = do not deliver; 1 = deliver)
30:12	Reserved
31	Valid

  - The **vector** (bits 7:0) determines which entry in the IDT is used or which other event is injected.
  - The **interruption 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).[2]

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).

The type **other event** is used for injection of events that are not delivered through the IDT.[1]

— For exceptions, the **deliver-error-code bit** (bit 11) determines whether delivery pushes an error code on the guest stack.

— 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 27.2).

- **VM-entry 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 VM-entry interruption-information field.
- **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.

See Section 26.6 for details regarding the mechanics of event injection, including the use of the interruption type and the VM-entry instruction length.

VM exits clear the valid bit (bit 31) in the VM-entry interruption-information field.

24.9 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 30).[2]

24.9.1 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 24-17.

Table 24-17. Format of Exit Reason

Bit Position(s)	Contents
15:0	Basic exit reason
16	Always cleared to 0
26:17	Not currently defined
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 27 is set to 1 if the VM exit occurred while the logical processor was in enclave mode.

2. The type hardware exception should be used for all other debug exceptions.

1. 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*.

2. Software can discover whether these fields can be written by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

A VM exit also sets this bit if it is incident to delivery of an event injected by VM entry and the guest interruptibility-state field indicates an enclave interrupt (bit 4 of the field is 1). See Section 27.2.1 for details.

— Bit 28 is set only by an SMM VM exit (see Section 31.15.2) that took priority over an MTF VM exit (see Section 25.5.2) that would have occurred had the SMM VM exit not occurred. See Section 31.15.2.3.

— 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 31.15.2.

— Because some VM-entry failures load processor state from the host-state area (see Section 26.8), 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 (SIPIs); task switches; INVEPT; INVLPG;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 27.2.1 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 27.2.1 and Section 31.15.2.3 for details of when and how this field is used.
- **Guest-physical address** (64 bits). This field is used VM exits due to EPT violations and EPT misconfigurations. See Section 27.2.1 for details of when and how this field is used.

24.9.2 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, and UD2); 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:

- **VM-exit interruption information** (32 bits). This field receives basic information associated with the event causing the VM exit. Table 24-18 describes this field.

Table 24-18. Format of the VM-Exit Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 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
30:13	Not currently defined
31	Valid

- **VM-exit interruption 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.

Section 27.2.2 provides details of how these fields are saved on VM exits.

24.9.3 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.[1] This information is provided in the following fields:

- **IDT-vectoring information** (32 bits). This field receives basic information associated with the event that was being delivered when the VM exit occurred. Table 24-19 describes this field.

Table 24-19. Format of the IDT-Vectoring Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 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)
30:12	Not currently defined
31	Valid

- **IDT-vectoring error code** (32 bits). For VM exits the occur during delivery of hardware exceptions that would have delivered an error code on the stack, this field receives that error code.

See Section 27.2.4 provides details of how these fields are saved on VM exits.

24.9.4 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 27.2.5 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 27.2.5 for details.

1. This includes cases in which the event delivery was caused by event injection as part of VM entry; see Section 26.6.1.2.
2. This field is also used for VM exits that occur during the delivery of a software interrupt or software exception.
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).

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).

24.9.5 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.

24.10 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 24-1): 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 24.6.2).

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 26.1).
- 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 25.1.3).
  - 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 30.3).
  - 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 26.3.1.5).

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 24.11.1). Before modifying the shadow-VMCS indicator, software should execute VMCLEAR for the VMCS to ensure that it is not active.

24.11 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.

24.11.1 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).

Software should not modify the shadow-VMCS indicator (see Table 24-1) 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 24.11.2). 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 25 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.

24.11.2 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 30 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 24-20.

Table 24-20. 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

1. As noted in Section 24.1, execution of the VMPTRLD instruction makes a VMCS is 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.

Table 24-20. Structure of VMCS Component Encoding (Contd.)

Bit Position(s)	Contents
12	Reserved (must be 0)
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).

24.11.3 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 unitize 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 24.1). 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 24-1 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 24.11.1), 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.

24.11.4 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 24.11.1). 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 28.4.3); the virtual-APIC page (Section 29.1); the posted interrupt descriptor (Section 29.6); and the virtualization-exception information area (Section 25.5.7.2).

24.11.5 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 24.2) 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 24.11.1).

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. 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.
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.

21. Updates to Chapter 26, Volume 3C

Change bars and green text show changes to Chapter 26 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C:* System Programming Guide, Part 3.

Changes to this chapter: Update to Section 26.2.1.1, "VM-Execution Control Fields". Update to Section 26.2.1.2, "VM-Exit Control Fields". Architectural LBR and HLAT additions as necessary.

CHAPTER 26
VM ENTRIES

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 26.1).
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 26.2).
3. The following may be performed in parallel or in any order (Section 26.3):
   - 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 26.4).
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 25.5.4 and Section 26.5).
7. An event may be injected in the guest context (Section 26.6).

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 26.1 may generate ordinary faults (for example, an invalid-opcode exception). Such faults are delivered normally.
- Some of the checks in Section 26.1 and all the checks in Section 26.2 cause control to pass to the instruction following the VM-entry instruction. The failure is indicated by setting RFLAGS.ZF[1] (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 30 for the error numbers.
- The checks in Section 26.3 and Section 26.4 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 26.8 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 26.1 and Section 26.2 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 26.3 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 26.1, Section 26.2, and Section 26.3 and there is no failure in loading MSRs.

Section 31.15 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 31.15.4.

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.

26.1 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 24.10), 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 24-3)
   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 30 for the error numbers.

26.2 CHECKS ON VMX CONTROLS AND HOST-STATE AREA

If the checks in Section 26.1 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 30).

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 26.2.1 through Section 26.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

26.2.1 Checks on VMX Controls

This section identifies VM-entry checks on the VMX control fields.

26.2.1.1 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).

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.

- 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).
- If the "use I/O bitmaps" VM-execution control is 1, bits 11:0 of each I/O-bitmap address must be 0. Neither address should set any bits beyond the processor's physical-address width.[1,2]
- If the "use MSR bitmaps" VM-execution control is 1, bits 11:0 of the MSR-bitmap address must be 0. The address should not set any bits beyond the processor's physical-address width.[3]
- If the "use TPR shadow" VM-execution control is 1, the virtual-APIC address must satisfy the following checks:
  - Bits 11:0 of the address must be 0.
  - The address should not set any bits beyond the processor's physical-address width.[4]

  If all of the above checks are satisfied and the "use TPR shadow" VM-execution control is 1, bytes 3:1 of VTPR (see Section 29.1.1) 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 "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.[5]
- 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 29.1.1).
- 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 "virtualize APIC-accesses" VM-execution control is 1, the APIC-access address must satisfy the following checks:
  - Bits 11:0 of the address must be 0.

---

1. 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.
2. If IA32_VMX_BASIC[48] is read as 1, these addresses must not set any bits in the range 63:32; see Appendix A.1.
3. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
4. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
5. "Virtual-interrupt delivery" 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 "virtual-interrupt delivery" VM-execution control were 0. See Section 24.6.2.

  - The address should not set any bits beyond the processor's physical-address width.[1]
- 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", and "virtual-interrupt delivery".[2]
- 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:[3]
  - 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 does not set any bits beyond the processor's physical-address width.[4]
- If the "enable VPID" VM-execution control is 1, the value of the VPID VM-execution control field must not be 0000H.[5]
- If the "enable EPT" VM-execution control is 1, the EPTP VM-execution control field (see Table 24-9 in Section 24.6.11) must satisfy the following checks:[6]
  - 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 (1 less than the EPT page-walk length) must be 3, indicating an EPT page-walk length of 4; see Section 28.3.2.
  - 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 and 63:N (where N is the processor's physical-address width) must all be 0.
- 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."[7]
- If the "enable PML" VM-execution control is 1, the PML address must satisfy the following checks:[8]
  - Bits 11:0 of the address must be 0.
  - The address should not set any bits beyond the processor's physical-address width.

1. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
2. "Virtualize x2APIC mode" and "APIC-register virtualization" 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 24.6.2.
3. "Process posted interrupts" 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 "process posted interrupts" VM-execution control were 0. See Section 24.6.2.
4. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
5. "Enable VPID" 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 VPID" VM-execution control were 0. See Section 24.6.2.
6. "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 24.6.2.
7. Some of these controls 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 both these controls were 0. Others are tertiary processor-based VM-execution controls. If bit 17 of the primary processor-based VM-execution controls is 0, VM entry functions as if both these controls were 0. See Section 24.6.2.
8. "Enable PML" 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 this control were 0. See Section 24.6.2.

- If the "sub-page write permissions for EPT" VM-execution control is 1, the SPPTP VM-execution control field (see Table 24-11 in Section 24.6.21) must satisfy the following checks:[1]
  - Bits 11:0 of the address must be 0.
  - The address should not set any bits beyond the processor's physical-address width.
- If the "enable VM functions" processor-based VM-execution control is 1, reserved bits in the VM-function controls must be clear.[2] 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 24.6.14):
  - If "EPTP switching" VM-function control is 1, the "enable EPT" VM-execution control must also be 1. In addition, the EPTP-list address must satisfy the following checks:
    - Bits 11:0 of the address must be 0.
    - The address must not set any bits beyond the processor's physical-address width.

  If the "enable VM functions" processor-based VM-execution control is 0, no checks are performed on the VM-function controls.
- If the "VMCS shadowing" VM-execution control is 1, the VMREAD-bitmap and VMWRITE-bitmap addresses must each satisfy the following checks:[3]
  - Bits 11:0 of the address must be 0.
  - The address must not set any bits beyond the processor's physical-address width.
- If the "EPT-violation #VE" VM-execution control is 1, the virtualization-exception information address must satisfy the following checks:[4]
  - Bits 11:0 of the address must be 0.
  - The address must not set any bits beyond the processor's physical-address width.
- 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.[5]
- If the "use TSC scaling" VM-execution control is 1, the TSC-multiplier must not be zero.[6]
- If the "enable HLAT" VM-execution control is 1, the following bits in the HLATP VM-execution control field (see Table 24-12 in Section 24.6.22) must be zero: bits 2:0, bits 11:5, and bits beyond the processor's physical-address width.[7]

1. "Sub-page write permissions for 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 this control were 0. See Section 24.6.2.
2. "Enable VM functions" 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 VM functions" VM-execution control were 0. See Section 24.6.2.
3. "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 24.6.2.
4. "EPT-violation #VE" 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 "EPT-violation #VE" VM-execution control were 0. See Section 24.6.2.
5. "Intel PT uses guest physical addresses" 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 "Intel PT uses guest physical addresses" VM-execution control were 0. See Section 24.6.2.
6. "Use TSC scaling" 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 "use TSC scaling" VM-execution control were 0. See Section 24.6.2.
7. "Enable HLAT" is a tertiary processor-based VM-execution control. If bit 17 of the primary processor-based VM-execution controls is 0, VM entry functions as if this control were 0. See Section 24.6.2.

26.2.1.2 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 should not set any bits beyond the processor's physical-address width.[1]
  - The address of the last byte in the VM-exit MSR-store area should not set any bits beyond the processor's physical-address width. 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.)

  If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1.
- 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 should not set any bits beyond the processor's physical-address width.
  - The address of the last byte in the VM-exit MSR-load area should not set any bits beyond the processor's physical-address width. 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.)

  If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1.

26.2.1.3 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 VM-entry interruption-information field (see Table 24-16 in Section 24.8.3), the VM-entry exception error code, and the VM-entry instruction length. If the valid bit (bit 31) in the VM-entry interruption-information field is 1, the following must hold:
  - The field's interruption 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 do not support the 1-setting of the "monitor trap flag" VM-execution control.
  - The field's vector (bits 7:0) is consistent with the interruption type:
    - If the interruption type is non-maskable interrupt (NMI), the vector is 2.
    - If the interruption type is hardware exception, the vector is at most 31.
    - If the interruption type is other event, the vector is 0 (pending MTF VM exit).

1. 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.

  - The field's deliver-error-code bit (bit 11) is 1 if each of the following holds: (1) the interruption type is hardware exception; (2) 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 interruption type is not hardware exception; (2) 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.
  - Reserved bits in the field (30:12) are 0.
  - If the deliver-error-code bit (bit 11) is 1, bits 31:16 of the VM-entry exception error-code field are 0.
  - If the interruption type is software interrupt, software exception, or privileged software exception, the VM-entry instruction-length field is in the range 0–15. 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 should not set any bits beyond the processor's physical-address width.[1]
  - The address of the last byte in the VM-entry MSR-load area should not set any bits beyond the processor's physical-address width. 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.
- 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.

26.2.2 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 23.8).[2]
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 23.8).
- 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 63:52 and bits in the range 51:32 beyond the processor's physical-address width must be 0.[3,4]
- 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 19-3).
- 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-).

---

1. 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.
2. 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 27.5.1.
3. 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.
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.

- 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.

26.2.3 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.

26.2.4 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.
  - 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.

26.3 CHECKING AND LOADING GUEST STATE

If all checks on the VMX controls and the host-state area pass (see Section 26.2), 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 26.8.

26.3.1 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.

26.3.1.1 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 23.8). 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 26.3.2.1.
- 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 23.8).
- 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.
  - The CR3 field must be such that bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width are 0.[4,5]

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 24.6.2.
2. If the capability MSR IA32_VMX_CR0_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_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.

  - 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 19-3).
- 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:
  - 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 32-6).
- 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.

26.3.1.2 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.)
- 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.

4. 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.
5. Bit 63 of the CR3 field in the guest-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.
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.

  - 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.[1]
- 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.

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 24.6.2.

— 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]

- 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) in the access-rights field 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.

26.3.1.3 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.

26.3.1.4 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 VM-entry interruption-information field is 1 and the interruption 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.

26.3.1.5 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 24.4.2). 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 VM-entry interruption-information field is 1, the interruption to be delivered (as defined by interruption type and vector) must not be one that would normally be blocked while a logical processor is in the activity state corresponding to the contents of the activity-state field. The following

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_CR0_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 24.4.1, SS.DPL corresponds to the logical processor's current privilege level (CPL).

items enumerate the interruptions (as specified in the VM-entry interruption-information field) whose injection is allowed for the different activity states:

- Active. Any interruption is allowed.
- HLT. The only events allowed are the following:
  - Those with interruption type external interrupt or non-maskable interrupt (NMI).
  - Those with interruption type hardware exception and vector 1 (debug exception) or vector 18 (machine-check exception).
  - Those with interruption type other event and vector 0 (pending MTF VM exit).

  See Table 24-16 in Section 24.8.3 for details regarding the format of the VM-entry interruption-information field.
- Shutdown. Only NMIs and machine-check exceptions are allowed.
- Wait-for-SIPI. No interruptions 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 VM-entry interruption-information field is 1 and the interruption 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 VM-entry interruption-information field is 1, and the interruption 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.(EAX=07H,ECX=0):EBX.SGX[bit 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 VM-entry interruption-information field is 1 and the interruption 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.(EAX=07H,ECX=0):EBX[bit 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.
  — Bits beyond the processor's physical-address width must be 0.[1,2]
  — 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 24.2).[3]
    - Bit 31 must contain the setting of the "VMCS shadowing" VM-execution control.[4] This implies that the referenced VMCS is a shadow VMCS (see Section 24.10) 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.

26.3.1.6 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 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).[5] 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.[6]
- 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 24.4.2).

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.[7] 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. 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.
2. If IA32_VMX_BASIC[48] is read as 1, this field must not set any bits in the range 63:32; see Appendix A.1.
3. 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.
4. "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 24.6.2.
5. 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.
6. "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 24.6.2.
7. 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.

26.3.2 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 26.3.1).

The loading of guest state is detailed in Section 26.3.2.1 to Section 26.3.2.4. 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 26.4). This loading occurs only after the state loading described in this section and the checking of VMCS contents described in Section 26.3.1.

26.3.2.1 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 26.3.2.2).
    - 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.

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 26.4.

- The SMBASE register is unmodified by all VM entries except those that return from SMM.

26.3.2.2 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 26.3.1.2). 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, the base address for LDTR is set to an undefined but canonical value.

— 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.

26.3.2.3 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 26.3.1.4, 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.)

26.3.2.4 Loading Page-Directory-Pointer-Table Entries

As noted in Section 26.3.1.6, 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 PDPTEs into internal, non-architectural registers based on the setting of the "enable EPT" VM-execution control:

- If the control is 0, the PDPTEs 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 26.3.2.1). The values loaded are treated as physical addresses in VMX non-root operation.
- If the control is 1, the PDPTEs are loaded from corresponding fields in the guest-state area (see Section 24.4.2). The values loaded are treated as guest-physical addresses in VMX non-root operation.

26.3.2.5 Updating Non-Register State

Section 28.4 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 EP4TA values (EP4TA 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 24.4.2). After doing so, the logical processor first causes PPR virtualization (Section 29.1.3) and then evaluates pending virtual interrupts (Section 29.2.1).

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 26.7.5. See Section 29.2.2 for details regarding the delivery of virtual interrupts.

26.3.3 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 8.10.4 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.

26.4 LOADING MSRS

VM entries may load MSRs from the VM-entry MSR-load area (see Section 24.8.2). 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 26.8.

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.

26.5 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 25.5.4.2. 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 25.5.4, 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 26.6), even if the valid bit in the VM-entry interruption-information field is 1. Such VM exits save the contents of VM-entry interruption-information and VM-entry exception error code fields into the IDT-vectoring information and IDT-vectoring 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.

26.6 EVENT INJECTION

If the valid bit in the VM-entry interruption-information field (see Section 24.8.3) is 1, VM entry causes an event to be delivered (or 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 interruption 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 26.6.1.
- If the interruption 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 26.6.2.

26.6.1 Vectored-Event Injection

VM entry delivers an injected vectored 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] 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.

Section 26.6.1.1 provides details of vectored-event injection. In general, the event is delivered exactly as if it had been generated normally.

If event delivery 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 delivered normally. If the nested exception is benign, it is delivered through the IDT. 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 6, "Interrupt 8—Double Fault Exception (#DF)" in *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 26.6.1.2 details cases in which event injection causes a VM exit.

26.6.1.1 Details of Vectored-Event Injection

The event-injection process is controlled by the contents of the VM-entry interruption information field (format given in Table 24-16), the VM-entry exception error-code field, 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 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 through the IDT.
- 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:[3]
  - — If VM entry successfully injects (with no nested exception) an event with interruption type external interrupt, NMI, or hardware exception, the current guest RIP is pushed on the stack.

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 26.6.1.1.
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.
3. While these items refer to RIP, the width of the value pushed (16 bits, 32 bits, or 64 bits) is determined normally.

  - If VM entry successfully injects (with no nested exception) an event with interruption 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.
  - 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 VM-entry interruption-information field, the contents of the VM-entry exception error-code field is pushed on the stack as an error code would be pushed during delivery of an exception.
- 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 interruption type external interrupt, NMI, hardware exception, and privileged software exception, or with interruption 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.
- 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 interruption 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 or a software exception 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).

26.6.1.2 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 VM-entry interruption-information 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 25.4.2).
- If event delivery encounters a nested exception, a VM exit may occur depending on the contents of the exception bitmap (see Section 25.2).
- 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 25.2).
- 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 25.2).
- 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 29.4 and may cause a VM exit.[1]

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 IDT-vectoring information field (see Section 27.2.4).

26.6.1.3 Event Injection for VM Entries to Real-Address Mode

If VM entry is loading CR0.PE with 0, any injected vectored event is delivered as would normally be done in real-address mode.[2] Specifically, VM entry uses the vector provided in the VM-entry interruption-information 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 15.1.4 in Volume 3A of the *IA-32 Intel® Architecture Software Developer's Manual*.

Because bit 11 (deliver error code) in the VM-entry interruption-information field must be 0 if CR0.PE will be 0 after VM entry (see Section 26.2.1.3), vectored 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 26.6.1.2.

26.6.2 Injection of Pending MTF VM Exits

If the interruption type in the VM-entry interruption-information 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 25.5.2 for the treatment of pending MTF VM exits.

26.7 SPECIAL FEATURES OF VM ENTRY

This section details a variety of features of VM entry. It uses the following terminology: a VM entry is **vectoring** if the valid bit (bit 31) of the VM-entry interruption information field is 1 and the interruption type in the field is 0

1. "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 24.6.2.
2. If the capability MSR IA32_VMX_CR0_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.

(external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception).

26.7.1 Interruptibility State

The interruptibility-state field in the guest-state area (see Table 24-3) 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 vectoring, 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 vectoring, 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 26.7.3).
  - 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 26.7.3. 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 26.6.1.1). 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 26.2.1.1.
- 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.

26.7.2 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 24.4.2. The use of this field is determined as follows:

- If the VM entry is vectoring, the logical processor is in the active state after VM entry. While the consistency checks described in Section 26.3.1.5 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 vectoring, 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. The error code used is 0000H, indicating "legacy shutdown." See *Intel® Trusted Execution Technology Preliminary Architecture Specification*.

- 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.

26.7.3 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 24.4.2). 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 vectoring with one of the following interruption 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 vectoring with either of the following interruption type: software interrupt or software exception.
- The VM entry is not vectoring 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 vectoring, 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.
- If the VM entry is vectoring (with interruption 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 6.9 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 26.7.7) and pending MTF VM exits (see Section 26.7.8. The exception takes priority

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

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.

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.

26.7.4 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 26.7.3) 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 25.5.1 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.

26.7.5 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 25.2 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 26.3.2.5 and Section 29.2.1).

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 26.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

26.7.6 NMI-Window Exiting

The "NMI-window exiting" VM-execution control may cause a VM exit to occur immediately after VM entry (see Section 25.2 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 26.7.3) 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 26.7.2). They do not occur if the logical processor just entered the wait-for-SIPI state.

26.7.7 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 29.1.1).[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 26.7.5, Section 26.7.6, and Section 26.7.8, 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 26.7.2). 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."

26.7.8 Pending MTF VM Exits

As noted in Section 26.6.2, 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 26.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

26.7.9 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.

26.8 VM-ENTRY FAILURES DURING OR AFTER LOADING GUEST STATE

VM-entry failures due to the checks identified in Section 26.3.1 and failures during the MSR loading identified in Section 26.4 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 26.3.1.

       34. VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs (see Section 26.4).

       41. VM-entry failure due to machine-check event. A machine-check event occurred during VM entry (see Section 26.9).
     - Bit 31 is set to 1 to indicate a VM-entry failure.
     - The remainder of the field (bits 30:16) is cleared.

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 24.6.2.

— 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 26.3.1.6).
  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 26.3.1.5).

  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 27.5). If this results in [CR4.PAE & CR0.PG & ~IA32_EFER.LMA] = 1, page-directory-pointer-table entries (PDPTEs) may be checked and loaded (see Section 27.5.4).
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 27.6).

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 VM-entry interruption-information field is not cleared.
- The guest-state area is not modified.
- No MSRs are saved into the VM-exit MSR-store area.

26.9 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 through the IDT.
- 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.

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 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

  - 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 through the guest IDT.
    - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- A VM-entry failure occurs as described in Section 26.8. 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.

22. Updates to Chapter 27, Volume 3C

Change bars and green 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: Update to Table 27-7, "Exit Qualification for EPT Violations ". Update to Section 27.3.1, "Saving Control Registers, Debug Registers, and MSRs".

CHAPTER 27
VM EXITS

VM exits occur in response to certain instructions and events in VMX non-root operation as detailed in Section 25.1 through Section 25.2. 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 27.2.
2. Processor state is saved in the guest-state area (Section 27.3).
3. MSRs may be saved in the VM-exit MSR-store area (Section 27.4). 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 27.5):
   — 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 31.15.6 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 27.6). 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 27.1 clarifies the nature of the architectural state before a VM exit begins. The steps described above are detailed in Section 27.2 through Section 27.6.

Section 31.15 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 31.15.2.

27.1 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 through the IDT. 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 29.4), EPT violation, EPT misconfiguration, page-modification log-full event (see Section 28.3.6), or SPP-related event (see Section 28.3.4) 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.NT[1]; 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 24.4.2) 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 25.5.2 and Section 26.7.8) 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 through the IDT (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 VM-exit interruption-information field; see Section 27.2.3.
- 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 VM-exit interruption-information field; see Section 27.2.3.
- 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 through the IDT. 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_MCG_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 29.4), 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 27.3.4).
- 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 29.3). (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 29.5.
  - — VM exits caused by APIC-write emulation (see Section 29.4.3.2) 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 24-3). 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 36, "Enclave Exiting Events"). An AEX modifies architectural state (Section 36.3). 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).

27.2 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 27.2.1 to Section 27.2.5 detail the use of these fields.

In addition to updating the VM-exit information fields, the valid bit (bit 31) is cleared in the VM-entry interruption-information 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]

27.2.1 Basic VM-Exit Information

Section 24.9.1 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.
  - 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 is incident to delivery of an event injected by VM entry and the guest interruptibility-state field indicates an enclave interruption (bit 4 of the field is 1).
  - The remainder of the field (bits 31:28 and bits 26:16) is cleared to 0 (certain SMM VM exits may set some of these bits; see Section 31.15.2.3).[2]
- **Exit qualification.** This field is saved for VM exits due to the following causes: debug exceptions; page-fault exceptions; start-up IPIs (SIPIs); system-management interrupts (SMIs) that arrive immediately after the execution of I/O instructions; task switches; INVEPT; INVLPG; INVPCID; INVVPID; LGDT; LIDT; LLDT; LTR; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; WBINVD; WBNOINVD; XRSTORS; XSAVES; control-register accesses; MOV DR; I/O instructions; MWAIT; accesses to the APIC-access page (see Section 29.4); EPT violations (see Section 28.3.3.2); EOI virtualization (see Section 29.1.4); APIC-write emulation (see Section 29.4.3.3); page-modification log full (see Section 28.3.6); and SPP-related events (see Section 28.3.4). For all other VM exits, this field is cleared. The following items provide details:
  - For a debug exception, the exit qualification contains information about the debug exception. The information has the format given in Table 27-1.

Table 27-1. 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.
12:4	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).

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 26.8.

Table 27-1. Exit Qualification for Debug Exceptions (Contd.)

Bit Position(s)	Contents
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 16.3.7, "RTM-Enabled Debugger Support," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*).[1]
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 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. 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 27-2.

— 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.

Table 27-2. 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.

— 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 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 24.9.4 and Section 27.2.5) 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 27-3.

— For MOV DR, the exit qualification contains information about the instruction and has the format given in Table 27-4.

— For an I/O instruction, the exit qualification contains information about the instruction and has the format given in Table 27-5.

— 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).

— 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.

— For an APIC-access VM exit resulting from a linear access or a guest-physical access to the APIC-access page (see Section 29.4), the exit qualification contains information about the access and has the format given in Table 27-6.[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 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 27-3. 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

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.

Table 27-3. Exit Qualification for Control-Register Accesses (Contd.)

Bit Positions	Contents
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 for IDT-vectoring information field (see Section 27.2.4) 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 29.4.4 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 29.4.6), 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 27-7.

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]

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).

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 27-4. 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.

Table 27-5. 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.

Bit 12 reports "NMI unblocking due to IRET"; see Section 27.2.3.

Table 27-6. 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 25.5.4) and accesses related to PEBS on processors with the "EPT-friendly" enhancement (see Section 19.9.5).
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

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 25.5.4) and accesses related to PEBS on processors with the "EPT-friendly" enhancement (see Section 19.9.5).

— For VM exits caused as part of EOI virtualization (Section 29.1.4), 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 29.4.3.3), 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 28.3.6), bit 12 of the exit qualification reports "NMI unblocking due to IRET." Bit 16 is set if the VM exit occurs during TAPT or EPT-friendly PEBS. All other bits of the exit qualification are undefined.

— For a VM exit due to an SPP-related event (Section 28.3.4), 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." Bit 16 is set if the VM exit occurs during TAPT or EPT-friendly PEBS. 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.

  — 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.

  — VM exits due to EPT violations that set bit 7 of the exit qualification (see Table 27-7; 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-

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.

Table 27-7. 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).
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. 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.
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 25.5.4).
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,[3] 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,[3] 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,[3] 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 27.2.3).
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.

Table 27-7. Exit Qualification for EPT Violations (Contd.)

Bit Position(s)	Contents
15	This bit is set if the EPT violation was caused as a result of guest-paging verification. See Section 28.3.3.2.
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 25.5.4.) 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 28.3.3.2). 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 any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present (see Section 28.3.2).
3. 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).

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 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.

— For all other VM exits, the field is undefined.

- **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.

27.2.2 Information for VM Exits Due to Vectored Events

Section 24.9.2 defines fields containing information for VM exits due to the following events: exceptions (including those generated by the instructions INT1, INT3, INTO, BOUND, UD0, UD1, and UD2); 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:

- **VM-exit interruption information** (format given in Table 24-18). 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.)

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.

- 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, and UD2 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. This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).[1] If bit 11 is set to 1, the error code is placed in the VM-exit interruption error code (see below).

— Bit 12 reports "NMI unblocking due to IRET"; see Section 27.2.3. The value of this bit is undefined if the VM exit is due to a double fault (the interruption type is hardware exception and the vector is 8).

— Bits 30:13 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.

- VM-exit interruption error code.

— For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the VM-exit interruption-information 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 through the IDT. 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 IDT-vectoring information 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.

27.2.3 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, or SPP-related events.

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 IDT-vectoring information field (see Section 27.2.4); 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 24-3) 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.

For VM exits due to faults, NMI unblocking due to IRET is saved in bit 12 of the VM-exit interruption-information field (Section 27.2.2). For VM exits due to EPT violations, page-modification log-full events, and SPP-related events, NMI unblocking due to IRET is saved in bit 12 of the exit qualification (Section 27.2.1).

1. 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.

(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.)

27.2.4 Information for VM Exits During Event Delivery

Section 24.9.3 defined fields containing information for VM exits that occur while delivering an event through the IDT and as a result of any of the following cases:[1]

- A fault occurs during event delivery and causes a VM exit (because the bit associated with the fault 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 25.4.2).
- Event delivery causes an APIC-access VM exit (see Section 29.4).
- An EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that occurs during event delivery.

These fields are used for VM exits that occur during delivery of events injected as part of VM entry (see Section 26.6.1.2).

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 following items detail the use of these fields:

- IDT-vectoring information (format given in Table 24-19). 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]

    - 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, and UD2 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. This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).[3] If bit 11 is set to 1, the error code is placed in the IDT-vectoring error code (see below).

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).
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.

— Bit 12 is undefined.

— Bits 30:13 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.

- IDT-vectoring error code.
  — For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the IDT-vectoring information field, this field receives the error code that would have been pushed on the stack by the event that was being delivered through the IDT 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.

27.2.5 Information for VM Exits Due to Instruction Execution

Section 24.9.4 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 25.1.2) or based on the settings of VM-execution controls (see Section 25.1.3): 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, RDMSR, RDPMC, RDRAND, RDSEED, RDTSC, RDTSCP, RSM, 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, or software exception.
  — 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 IDT-vectoring information 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 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 25.5.6.2).

---

3. 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.

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 29.4.6) even if encountered during delivery of a software interrupt, privileged software exception, or software exception.

- EAX = 0 and either ECX ≥ 512 or the value of ECX selects an invalid tentative EPTP value (see Section 25.5.6.3).

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 26.6.1.2). 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 27-8. 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.
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, 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 27-8.[1]
  - For VM exits due to attempts to execute INVEPT, INVPCID, or INVVPID, the field has the format is given in Table 27-9.
  - For VM exits due to attempts to execute LIDT, LGDT, SIDT, or SGDT, the field has the format is given in Table 27-10.
  - For VM exits due to attempts to execute LLDT, LTR, SLDT, or STR, the field has the format is given in Table 27-11.
  - For VM exits due to attempts to execute RDRAND, RDSEED, TPAUSE, or UMWAIT, the field has the format is given in Table 27-12.
  - For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, or XSAVES, the field has the format is given in Table 27-13.

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 27-8 is used by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

— For VM exits due to attempts to execute VMREAD or VMWRITE, the field has the format is given in Table 27-14.

— For VM exits due to attempts to execute LOADIWKEY, the field has the format is given in Table 27-15.

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 31.15.2.3. Note that, if the VM exit occurred in enclave mode, these fields are all cleared.

Table 27-9. 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 27-10. 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).
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

Table 27-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT (Contd.)

Bit Position(s)	Content
31:30	Undefined.

Table 27-11. 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).

Table 27-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR (Contd.)

Bit Position(s)	Content
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
29:28	Instruction identity: 0: SLDT 1: STR 2: LLDT 3: LTR
31:30	Undefined.

Table 27-12. Format of the VM-Exit Instruction-Information Field as Used for RDRAND, RDSEED, TPAUSE, and UMWAIT

Bit Position(s)	Content
2:0	Undefined.
6:3	Operand register (destination for RDRAND and RDSEED; source for TPAUSE and UMWAIT): 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 27-13. 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.

Table 27-13. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, and XSAVES (Contd.)

Bit Position(s)	Content
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.

Table 27-14. 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.

Table 27-14. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE (Contd.)

Bit Position(s)	Content
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)

Table 27-15. 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).

27.3 SAVING GUEST STATE

VM exits save certain components of processor state into corresponding fields in the guest-state area of the VMCS (see Section 24.4). On processors that support Intel 64 architecture, the full value of each natural-width field (see Section 24.11.2) 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 27.1 for a discussion of which architectural updates occur at that time.

Section 27.3.1 through Section 27.3.4 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.

27.3.1 Saving Control Registers, Debug Registers, and MSRs

Contents of certain control registers, debug registers, and MSRs is 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.
- The value of the SMBASE field is undefined after all VM exits except SMM VM exits. See Section 31.15.2.

27.3.2 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 24.4.1) 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.
  - LDTR. The value saved for the base address is always canonical.
- 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.

27.3.3 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 27.2.2), 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 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.

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.

  - 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 29.1.1) below that of TPR threshold VM-execution control field (see Section 29.1.2), 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 29.4.3.2) 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 through the IDT, 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 gate[1] or into the old task-state segment had the event been delivered through a task gate) had the event been delivered through the IDT. 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 through the IDT:
    - If the VM exit stored 0 for bit 31 for IDT-vectoring information field (because the VM exit did not occur during delivery of an event through the IDT; see Section 27.2.4), the value saved is 1.
    - If the VM exit stored 1 for bit 31 for IDT-vectoring information field (because the VM exit did occur during delivery of an event through the IDT), the value saved is the value that would have appeared in the saved RFLAGS image had the event been delivered through the IDT (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.

27.3.4 Saving Non-Register State

Information corresponding to guest non-register state is saved as follows:

- The activity-state field is saved with the logical processor's activity state before the VM exit.[3] See Section 27.1 for details of how events leading to a VM exit may affect the activity state.
- The interruptibility-state field is saved to reflect the logical processor's interruptibility before the VM exit.
  - See Section 27.1 for details of how events leading to a VM exit may affect this state.

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.
3. 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.

— 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 that is incident to delivery of an event injected by VM entry leaves this bit unmodified.

- 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",[1] "virtualized EOI", "APIC write", or "monitor trap flag."

— A VM exit due to trace-address pre-translation (TAPT; see Section 25.5.4) or due to accesses related to PEBS on processors with the "EPT-friendly" enhancement (see Section 19.9.5). 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 26.7.3). 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 26.7.3) 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 16.3.7, "RTM-Enabled Debugger 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.")

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.")

1. This item includes VM exits that occur as a result of certain VM entries (Section 26.7.7).

  - 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 26.7.3). 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 26.7.3) 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 25.5.1). 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.

27.4 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 24.7.2). 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.

1. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 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 24.6.2.

- 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 27.7.

27.5 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 27.5.1 to Section 27.5.5. 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 27.7.

In addition to loading host state, VM exits clear address-range monitoring (Section 27.5.6).

After the state loading described in this section, VM exits may load MSRs from the VM-exit MSR-load area (see Section 27.6). This loading occurs only after the state loading described in this section.

27.5.1 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 23.8).[1]
    - For CR3, bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width (they are cleared to 0).[2] (This item applies only to processors that support Intel 64 architecture.)
    - For CR4, any bits that are fixed in VMX operation (see Section 23.8).
  - 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.
- DR7 is set to 400H.
- 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.

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.
2. 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.

— 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.[1]

— 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 27.5.2).
- 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.

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.

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 27.6.

27.5.2 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 Section 26.3.1.2 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.

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.

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 FFFFFH 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.
  - 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 27.5.1), 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 (although the base address is always canonical).

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.

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.

27.5.3 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.

27.5.4 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 4.4 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 27.7). 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.

27.5.5 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 24-3). Other VM exits do not affect blocking by NMI. (See Section 27.1 for the case in which an NMI causes a VM exit indirectly.)
- There are no pending debug exceptions after a VM exit.

Section 28.4 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 EP4TA values (EP4TA 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.

27.5.6 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 8.10.4 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.

---

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.

27.6 LOADING MSRS

VM exits may load MSRs from the VM-exit MSR-load area (see Section 24.7.2). 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 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 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 27.7.

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.

27.7 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 24.2). The following values are used:

1. There was a failure in saving guest MSRs (see Section 27.4).
2. Host checking of the page-directory-pointer-table entries (PDPTEs) failed (see Section 27.5.4).
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 27.6).
5. There was a machine-check event during VM exit (see Section 27.8).
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 27.5).

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 27.5) 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

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.

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 *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).

27.8 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:[2]
    - 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 generated:
    - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered through the guest IDT.
    - 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 through the host IDT.
- A VMX abort is generated (see Section 27.7). 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.

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 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.
2. 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 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

23. Updates to Chapter 28, Volume 3C

Change bars and green 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: HLAT additions as necessary throughout chapter.

CHAPTER 28
VMX SUPPORT FOR ADDRESS TRANSLATION

The architecture for VMX operation includes two features that support address translation: virtual-processor identifiers (VPIDs) and the extended page-table mechanism (EPT). VPIDs are a mechanism for managing translations of linear addresses. EPT defines a layer of address translation that augments the translation of linear addresses.

Section 28.1 details the architecture of VPIDs. Section 28.3 provides the details of EPT. Section 28.4 explains how a logical processor may cache information from the paging structures, how it may use that cached information, and how software can managed the cached information.

28.1 VIRTUAL PROCESSOR IDENTIFIERS (VPIDS)

The original architecture for VMX operation required VMX transitions to flush the TLBs and paging-structure caches. This ensured that translations cached for the old linear-address space would not be used after the transition.

Virtual-processor identifiers (**VPIDs**) introduce to VMX operation a facility by which a logical processor may cache information for multiple linear-address spaces. When VPIDs are used, VMX transitions may retain cached information and the logical processor switches to a different linear-address space.

Section 28.4 details the mechanisms by which a logical processor manages information cached for multiple address spaces. A logical processor may tag some cached information with a 16-bit VPID. This section specifies how the current VPID is determined at any point in time:

- The current VPID is 0000H in the following situations:
  - Outside VMX operation. (This includes operation in system-management mode under the default treatment of SMIs and SMM with VMX operation; see Section 31.14.)
  - In VMX root operation.
  - In VMX non-root operation when the "enable VPID" VM-execution control is 0.
- If the logical processor is in VMX non-root operation and the "enable VPID" VM-execution control is 1, the current VPID is the value of the VPID VM-execution control field in the VMCS. (VM entry ensures that this value is never 0000H; see Section 26.2.1.1.)

VPIDs and PCIDs (see Section 4.10.1) can be used concurrently. When this is done, the processor associates cached information with both a VPID and a PCID. Such information is used only if the current VPID and PCID **both** match those associated with the cached information.

28.2 HYPERVISOR-MANAGED LINEAR-ADDRESS TRANSLATION (HLAT)

Hypervisor-managed linear-address translation (**HLAT**) is a feature that changes the way in which linear addresses are translated in VMX non-root operation. Instead of ordinary paging, translation uses a modified process called **HLAT paging**.

HLAT paging is used only if the "enable HLAT" VM-execution control is 1. HLAT paging is used only for the paging modes 4-level paging and 5-level paging. Because HLAT paging is a modification of ordinary paging, details of the feature are given in Section 4.5, which describes the operation of 4-level paging and 5-level paging.

28.3 THE EXTENDED PAGE TABLE MECHANISM (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 addresses that would normally be treated as physical addresses (and used to access memory) are instead treated as **guest-physical addresses**. Guest-physical addresses are translated by traversing a set of **EPT paging structures** to produce physical addresses that are used to access memory.

- Section 28.3.1 gives an overview of EPT.
- Section 28.3.2 describes operation of EPT-based address translation.
- Section 28.3.3 discusses VM exits that may be caused by EPT.
- Section 28.3.7 describes interactions between EPT and memory typing.

28.3.1 EPT Overview

EPT is used when the "enable EPT" VM-execution control is 1.[1] It translates the guest-physical addresses used in VMX non-root operation and those used by VM entry for event injection.

The translation from guest-physical addresses to physical addresses is determined by a set of **EPT paging structures**. The EPT paging structures are similar to those used to translate linear addresses while the processor is in IA-32e mode. Section 28.3.2 gives the details of the EPT paging structures.

If CR0.PG = 1, linear addresses are translated through paging structures referenced through control register CR3.[2] While the "enable EPT" VM-execution control is 1, these are called **guest paging structures**. There are no guest paging structures if CR0.PG = 0.[3]

When the "enable EPT" VM-execution control is 1, the identity of **guest-physical addresses** depends on the value of CR0.PG:

- If CR0.PG = 0, each linear address is treated as a guest-physical address.
- If CR0.PG = 1, guest-physical addresses are those derived from the contents of control register CR3 and the guest paging structures. (This includes the values of the PDPTEs, which logical processors store in internal, non-architectural registers.) The latter includes (in page-table entries and in other paging-structure entries for which bit 7—PS—is 1) the addresses to which linear addresses are translated by the guest paging structures.

If CR0.PG = 1, the translation of a linear address to a physical address requires multiple translations of guest-physical addresses using EPT. Assume, for example, that CR4.PAE = CR4.PSE = 0. The translation of a 32-bit linear address then operates as follows:

- Bits 31:22 of the linear address select an entry in the guest page directory located at the guest-physical address in CR3. The guest-physical address of the guest page-directory entry (PDE) is translated through EPT to determine the guest PDE's physical address.
- Bits 21:12 of the linear address select an entry in the guest page table located at the guest-physical address in the guest PDE. The guest-physical address of the guest page-table entry (PTE) is translated through EPT to determine the guest PTE's physical address.
- Bits 11:0 of the linear address is the offset in the page frame located at the guest-physical address in the guest PTE. The guest-physical address determined by this offset is translated through EPT to determine the physical address to which the original linear address translates.

In addition to translating a guest-physical address to a physical address, EPT specifies the privileges that software is allowed when accessing the address. Attempts at disallowed accesses are called **EPT violations** and cause VM exits. See Section 28.3.3.

A processor uses EPT to translate guest-physical addresses only when those addresses are used to access memory. This principle implies the following:

- The MOV to CR3 instruction loads CR3 with a guest-physical address. Whether that address is translated through EPT depends on whether PAE paging is being used.[4]

---

1. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, the logical processor operates as if the "enable EPT" VM-execution control were 0. See Section 24.6.2.
2. If HLAT paging is enabled, the processor uses the HLATP VMCS field instead of CR3. See Section 4.5.1. For simplicity, the remainder of this section refers only to CR3.
3. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG can be 0 in VMX non-root operation only if the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
4. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

— If PAE paging is not being used, the instruction does not use that address to access memory and does **not** cause it to be translated through EPT. (If CR0.PG = 1, the address will be translated through EPT on the next memory accessing using a linear address.)

— If PAE paging is being used, the instruction loads the four (4) page-directory-pointer-table entries (PDPTEs) from that address and it **does** cause the address to be translated through EPT.

- Section 4.4.1 identifies executions of MOV to CR0 and MOV to CR4 that load the PDPTEs from the guest-physical address in CR3. Such executions cause that address to be translated through EPT.
- The PDPTEs contain guest-physical addresses. The instructions that load the PDPTEs (see above) do not use those addresses to access memory and do **not** cause them to be translated through EPT. The address in a PDPTE will be translated through EPT on the next memory accessing using a linear address that uses that PDPTE.

28.3.2 EPT Translation Mechanism

The EPT translation mechanism uses only bits 47:0 of each guest-physical address.[1] It uses a page-walk length of 4, meaning that at most 4 EPT paging-structure entries are accessed to translate a guest-physical address.[2]

These 48 bits are partitioned by the logical processor to traverse the EPT paging structures:

- A 4-KByte naturally aligned EPT PML4 table is located at the physical address specified in bits 51:12 of the extended-page-table pointer (EPTP), a VM-execution control field (see Table 24-9 in Section 24.6.11). An EPT PML4 table comprises 512 64-bit entries (EPT PML4Es). An EPT PML4E is selected using the physical address defined as follows:

  — Bits 63:52 are all 0.

  — Bits 51:12 are from the EPTP.

  — Bits 11:3 are bits 47:39 of the guest-physical address.

  — Bits 2:0 are all 0.

  Because an EPT PML4E is identified using bits 47:39 of the guest-physical address, it controls access to a 512-GByte region of the guest-physical-address space. The format of an EPT PML4E is given in Table 28-1.

- A 4-KByte naturally aligned EPT page-directory-pointer table is located at the physical address specified in bits 51:12 of the EPT PML4E. An EPT page-directory-pointer table comprises 512 64-bit entries (EPT PDPTEs). An EPT PDPTE is selected using the physical address defined as follows:

  — Bits 63:52 are all 0.

  — Bits 51:12 are from the EPT PML4E.

  — Bits 11:3 are bits 38:30 of the guest-physical address.

  — Bits 2:0 are all 0.

Because an EPT PDPTE is identified using bits 47:30 of the guest-physical address, it controls access to a 1-GByte region of the guest-physical-address space. Use of the EPT PDPTE depends on the value of bit 7 in that entry:[3]

- If bit 7 of the EPT PDPTE is 1, the EPT PDPTE maps a 1-GByte page. The final physical address is computed as follows:

  — Bits 63:52 are all 0.

  — Bits 51:30 are from the EPT PDPTE.

---

1. No processors supporting the Intel 64 architecture support more than 48 physical-address bits. Thus, no such processor can produce a guest-physical address with more than 48 bits. An attempt to use such an address causes a page fault. An attempt to load CR3 with such an address causes a general-protection fault. If PAE paging is being used, an attempt to load CR3 that would load a PDPTE with such an address causes a general-protection fault.
2. Future processors may include support for other EPT page-walk lengths. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine what EPT page-walk lengths are supported.
3. Not all processors allow bit 7 of an EPT PDPTE to be set to 1. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether this is allowed.

Table 28-1. Format of an EPT PML4 Entry (PML4E) that References an EPT Page-Directory-Pointer Table

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 512-GByte region controlled by this entry
1	Write access; indicates whether writes are allowed to the 512-GByte region controlled by this entry
2	If the "mode-based execute control for EPT" VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 512-GByte region controlled by this entry If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 512-GByte region controlled by this entry
7:3	Reserved (must be 0)
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 512-GByte region controlled by this entry (see Section 28.3.5). Ignored if bit 6 of EPTP is 0
9	Ignored
10	Execute access for user-mode linear addresses. If the "mode-based execute control for EPT" VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 512-GByte region controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored
(N–1):12	Physical address of 4-KByte aligned EPT page-directory-pointer table referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

1. N is the physical-address width supported by the processor. 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.

— Bits 29:0 are from the original guest-physical address.

The format of an EPT PDPTE that maps a 1-GByte page is given in Table 28-2.

- If bit 7 of the EPT PDPTE is 0, a 4-KByte naturally aligned EPT page directory is located at the physical address specified in bits 51:12 of the EPT PDPTE. The format of an EPT PDPTE that references an EPT page directory is given in Table 28-3.

Table 28-2. Format of an EPT Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GByte Page

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 1-GByte page referenced by this entry
1	Write access; indicates whether writes are allowed to the 1-GByte page referenced by this entry
2	If the "mode-based execute control for EPT" VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 1-GByte page controlled by this entry If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 1-GByte page controlled by this entry
5:3	EPT memory type for this 1-GByte page (see Section 28.3.7)
6	Ignore PAT memory type for this 1-GByte page (see Section 28.3.7)
7	Must be 1 (otherwise, this entry references an EPT page directory)
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 1-GByte page referenced by this entry (see Section 28.3.5). Ignored if bit 6 of EPTP is 0
9	If bit 6 of EPTP is 1, dirty flag for EPT; indicates whether software has written to the 1-GByte page referenced by this entry (see Section 28.3.5). Ignored if bit 6 of EPTP is 0
10	Execute access for user-mode linear addresses. If the "mode-based execute control for EPT" VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 1-GByte page controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored
29:12	Reserved (must be 0)
(N–1):30	Physical address of the 1-GByte page referenced by this entry[1]
51:N	Reserved (must be 0)
56:52	Ignored
57	Verify guest paging. If the "guest-paging verification" VM-execution control is 1, indicates limits on the guest paging structures used to access the 1-GByte page controlled by this entry (see Section 28.3.3.2). If that control is 0, this bit is ignored.
58	Paging-write access. If the "EPT paging-write control" VM-execution control is 1, indicates that guest paging may update the 1-GByte page controlled by this entry (see Section 28.3.3.2). If that control is 0, this bit is ignored.
59	Ignored
60	Supervisor shadow stack. If bit 7 of EPTP is 1, indicates whether supervisor shadow stack accesses are allowed to guest-physical addresses in the 1-GByte page mapped by this entry (see Section 28.3.3.2). Ignored if bit 7 of EPTP is 0
62:61	Ignored
63	Suppress #VE. If the "EPT-violation #VE" VM-execution control is 1, EPT violations caused by accesses to this page are convertible to virtualization exceptions only if this bit is 0 (see Section 25.5.7.1). If "EPT-violation #VE" VM-execution control is 0, this bit is ignored.

NOTES:

1. N is the physical-address width supported by the logical processor.

Table 28-3. Format of an EPT Page-Directory-Pointer-Table Entry (PDPTE) that References an EPT Page Directory

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 1-GByte region controlled by this entry
1	Write access; indicates whether writes are allowed to the 1-GByte region controlled by this entry
2	If the "mode-based execute control for EPT" VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 1-GByte region controlled by this entry If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 1-GByte region controlled by this entry
7:3	Reserved (must be 0)
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 1-GByte region controlled by this entry (see Section 28.3.5). Ignored if bit 6 of EPTP is 0
9	Ignored
10	Execute access for user-mode linear addresses. If the "mode-based execute control for EPT" VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 1-GByte region controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored
(N–1):12	Physical address of 4-KByte aligned EPT page directory referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

1. N is the physical-address width supported by the logical processor.

An EPT page-directory comprises 512 64-bit entries (PDEs). An EPT PDE is selected using the physical address defined as follows:

— Bits 63:52 are all 0.

— Bits 51:12 are from the EPT PDPTE.

— Bits 11:3 are bits 29:21 of the guest-physical address.

— Bits 2:0 are all 0.

Because an EPT PDE is identified using bits 47:21 of the guest-physical address, it controls access to a 2-MByte region of the guest-physical-address space. Use of the EPT PDE depends on the value of bit 7 in that entry:

- If bit 7 of the EPT PDE is 1, the EPT PDE maps a 2-MByte page. The final physical address is computed as follows:

  — Bits 63:52 are all 0.

  — Bits 51:21 are from the EPT PDE.

  — Bits 20:0 are from the original guest-physical address.

  The format of an EPT PDE that maps a 2-MByte page is given in Table 28-4.

- If bit 7 of the EPT PDE is 0, a 4-KByte naturally aligned EPT page table is located at the physical address specified in bits 51:12 of the EPT PDE. The format of an EPT PDE that references an EPT page table is given in Table 28-5.

  An EPT page table comprises 512 64-bit entries (PTEs). An EPT PTE is selected using a physical address defined as follows:

  — Bits 63:52 are all 0.

Table 28-4. Format of an EPT Page-Directory Entry (PDE) that Maps a 2-MByte Page

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 2-MByte page referenced by this entry
1	Write access; indicates whether writes are allowed to the 2-MByte page referenced by this entry
2	If the "mode-based execute control for EPT" VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 2-MByte page controlled by this entry If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 2-MByte page controlled by this entry
5:3	EPT memory type for this 2-MByte page (see Section 28.3.7)
6	Ignore PAT memory type for this 2-MByte page (see Section 28.3.7)
7	Must be 1 (otherwise, this entry references an EPT page table)
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 28.3.5). Ignored if bit 6 of EPTP is 0
9	If bit 6 of EPTP is 1, dirty flag for EPT; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 28.3.5). Ignored if bit 6 of EPTP is 0
10	Execute access for user-mode linear addresses. If the "mode-based execute control for EPT" VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 2-MByte page controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored
20:12	Reserved (must be 0)
(N–1):21	Physical address of the 2-MByte page referenced by this entry[1]
51:N	Reserved (must be 0)
56:52	Ignored
57	Verify guest paging. If the "guest-paging verification" VM-execution control is 1, indicates limits on the guest paging structures used to access the 2-MByte page controlled by this entry (see Section 28.3.3.2). If that control is 0, this bit is ignored.
58	Paging-write access. If the "EPT paging-write control" VM-execution control is 1, indicates that guest paging may update the 2-MByte page controlled by this entry (see Section 28.3.3.2). If that control is 0, this bit is ignored.
59	Ignored
60	Supervisor shadow stack. If bit 7 of EPTP is 1, indicates whether supervisor shadow stack accesses are allowed to guest-physical addresses in the 2-MByte page mapped by this entry (see Section 28.3.3.2). Ignored if bit 7 of EPTP is 0
62:61	Ignored
63	Suppress #VE. If the "EPT-violation #VE" VM-execution control is 1, EPT violations caused by accesses to this page are convertible to virtualization exceptions only if this bit is 0 (see Section 25.5.7.1). If "EPT-violation #VE" VM-execution control is 0, this bit is ignored.

NOTES:

1. N is the physical-address width supported by the logical processor.

— Bits 51:12 are from the EPT PDE.

  - Bits 11:3 are bits 20:12 of the guest-physical address.
  - Bits 2:0 are all 0.
- Because an EPT PTE is identified using bits 47:12 of the guest-physical address, every EPT PTE maps a 4-KByte page. The final physical address is computed as follows:
  - Bits 63:52 are all 0.
  - Bits 51:12 are from the EPT PTE.
  - Bits 11:0 are from the original guest-physical address.

  The format of an EPT PTE is given in Table 28-6.

An EPT paging-structure entry is **present** if any of bits 2:0 is 1; otherwise, the entry is **not present**. The processor ignores bits 62:3 and uses the entry neither to reference another EPT paging-structure entry nor to produce a physical address. A reference using a guest-physical address whose translation encounters an EPT paging-structure that is not present causes an EPT violation (see Section 28.3.3.2). (If the "EPT-violation #VE" VM-execution control is 1, the EPT violation is convertible to a virtualization exception only if bit 63 is 0; see Section 25.5.7.1. If the "EPT-violation #VE" VM-execution control is 0, this bit is ignored.)

Table 28-5. Format of an EPT Page-Directory Entry (PDE) that References an EPT Page Table

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 2-MByte region controlled by this entry
1	Write access; indicates whether writes are allowed to the 2-MByte region controlled by this entry
2	If the "mode-based execute control for EPT" VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 2-MByte region controlled by this entry If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 2-MByte region controlled by this entry
6:3	Reserved (must be 0)
7	Must be 0 (otherwise, this entry maps a 2-MByte page)
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 2-MByte region controlled by this entry (see Section 28.3.5). Ignored if bit 6 of EPTP is 0
9	Ignored
10	Execute access for user-mode linear addresses. If the "mode-based execute control for EPT" VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 2-MByte region controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored
(N–1):12	Physical address of 4-KByte aligned EPT page table referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

1. N is the physical-address width supported by the logical processor.

NOTE

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. If bits 2:0 are all 0 but bit 10 is 1, the entry is used normally to reference another EPT paging-structure entry or to produce a physical address.

The discussion above describes how the EPT paging structures reference each other and how the logical processor traverses those structures when translating a guest-physical address. It does not cover all details of the translation process. Additional details are provided as follows:

- Situations in which the translation process may lead to VM exits (sometimes before the process completes) are described in Section 28.3.3.
- Interactions between the EPT translation mechanism and memory typing are described in Section 28.3.7.

Figure 28-1 gives a summary of the formats of the EPTP and the EPT paging-structure entries. For the EPT paging structure entries, it identifies separately the format of entries that map pages, those that reference other EPT paging structures, and those that do neither because they are not present; bits 2:0 and bit 7 are highlighted because they determine how a paging-structure entry is used. (Figure 28-1 does not comprehend the fact that, if the "mode-based execute control for EPT" VM-execution control is 1, an entry is present if any of bits 2:0 or bit 10 is 1.)

28.3.3 EPT-Induced VM Exits

Accesses using guest-physical addresses may cause VM exits due to EPT misconfigurations, EPT violations, and page-modification log-full events. An **EPT misconfiguration** occurs when, in the course of translating a guest-physical address, the logical processor encounters an EPT paging-structure entry that contains an unsupported value (see Section 28.3.3.1). An **EPT violation** occurs when there is no EPT misconfiguration but the EPT paging-structure entries disallow an access using the guest-physical address (see Section 28.3.3.2). A **page-modification log-full event** occurs when the logical processor determines a need to create a page-modification log entry and the current log is full (see Section 28.3.6).

These events occur only due to an attempt to access memory with a guest-physical address. Loading CR3 with a guest-physical address with the MOV to CR3 instruction can cause neither an EPT configuration nor an EPT violation until that address is used to access a paging structure.[1]

If the "EPT-violation #VE" VM-execution control is 1, certain EPT violations may cause virtualization exceptions instead of VM exits. See Section 25.5.7.1.

28.3.3.1 EPT Misconfigurations

An EPT misconfiguration occurs if translation of a guest-physical address encounters an EPT paging-structure entry that meets any of the following conditions:

- Bit 0 of the entry is clear (indicating that data reads are not allowed) and any of the following hold:
  - Bit 1 is set (indicating that data writes are allowed).
  - The processor does not support execute-only translations and either of the following hold:
    - Bit 2 is set (indicating that instruction fetches are allowed).[2]
    - The "mode-based execute control for EPT" VM-execution control is 1 and bit 10 is set (indicating that instruction fetches are allowed from user-mode linear addresses).

    Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP to determine whether execute-only translations are supported (see Appendix A.10).
  - The "EPT paging-write control" VM-execution control is 1, the entry maps a page, and bit 58 is set (indicating that paging writes are allowed).
- The entry is present (see Section 28.3.2) and of the following holds:

1. If the logical processor is using PAE paging—because CR0.PG = CR4.PAE = 1 and IA32_EFER.LMA = 0—the MOV to CR3 instruction loads the PDPTEs from memory using the guest-physical address being loaded into CR3. In this case, therefore, the MOV to CR3 instruction may cause an EPT misconfiguration, an EPT violation, or a page-modification log-full event.
2. If the "mode-based execute control for EPT" VM-execution control is 1, setting bit 2 indicates that instruction fetches are allowed from supervisor-mode linear addresses.

Table 28-6. Format of an EPT Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 4-KByte page referenced by this entry
1	Write access; indicates whether writes are allowed to the 4-KByte page referenced by this entry
2	If the "mode-based execute control for EPT" VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 4-KByte page controlled by this entry If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 4-KByte page controlled by this entry
5:3	EPT memory type for this 4-KByte page (see Section 28.3.7)
6	Ignore PAT memory type for this 4-KByte page (see Section 28.3.7)
7	Ignored
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 28.3.5). Ignored if bit 6 of EPTP is 0
9	If bit 6 of EPTP is 1, dirty flag for EPT; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 28.3.5). Ignored if bit 6 of EPTP is 0
10	Execute access for user-mode linear addresses. If the "mode-based execute control for EPT" VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 4-KByte page controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored
(N–1):12	Physical address of the 4-KByte page referenced by this entry[1]
51:N	Reserved (must be 0)
56:52	Ignored
57	Verify guest paging. If the "guest-paging verification" VM-execution control is 1, indicates limits on the guest paging structures used to access the 4-KByte page controlled by this entry (see Section 28.3.3.2). If that control is 0, this bit is ignored.
58	Paging-write access. If the "EPT paging-write control" VM-execution control is 1, indicates that guest paging may update the 4-KByte page controlled by this entry (see Section 28.3.3.2). If that control is 0, this bit is ignored.
59	Ignored
60	Supervisor shadow stack. If bit 7 of EPTP is 1, indicates whether supervisor shadow stack accesses are allowed to guest-physical addresses in the 4-KByte page mapped by this entry (see Section 28.3.3.2). Ignored if bit 7 of EPTP is 0
61	Sub-page write permissions. If the "sub-page write permissions for EPT" VM-execution control is 1, writes to individual 128-byte regions of the 4-KByte page referenced by this entry may be allowed even if the page would normally not be writable (see Section 28.3.4). If "sub-page write permissions for EPT" VM-execution control is 0, this bit is ignored.
62	Ignored
63	Suppress #VE. If the "EPT-violation #VE" VM-execution control is 1, EPT violations caused by accesses to this page are convertible to virtualization exceptions only if this bit is 0 (see Section 25.5.7.1). If "EPT-violation #VE" VM-execution control is 0, this bit is ignored.

NOTES:

1. N is the physical-address width supported by the logical processor.

— A reserved bit is set. This includes the setting of a bit in the range 51:12 that is beyond the logical processor's physical-address width.[1] See Section 28.3.2 for details of which bits are reserved in which EPT paging-structure entries.

— The entry is the last one used to translate a guest physical address (either an EPT PDE with bit 7 set to 1 or an EPT PTE) and the value of bits 5:3 (EPT memory type) is 2, 3, or 7 (these values are reserved).

EPT misconfigurations result when an EPT paging-structure entry is configured with settings reserved for future functionality. Software developers should be aware that such settings may be used in the future and that an EPT paging-structure entry that causes an EPT misconfiguration on one processor might not do so in the future.

28.3.3.2 EPT Violations

An EPT violation may occur during an access using a guest-physical address whose translation does not cause an EPT misconfiguration. An EPT violation occurs in any of the following situations:

- Translation of the guest-physical address encounters an EPT paging-structure entry that is not present (see Section 28.3.2).

1. 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.

- The access is a data read and, for any byte to be read, bit 0 (read access) was clear in any of the EPT paging-structure entries used to translate the guest-physical address of the byte. Reads by the logical processor of guest paging structures to translate a linear address are considered to be data reads.

63	62	61	60	59	58	57	56	55	54	53	52	51	M[1]	M-1	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved														Address of EPT PML4 table																								Rsvd.				SSS[2]	A/D	EPT PWL-1			EPT PS MT			**EPTP[3]**
Ignored													Rsvd.	Address of EPT page-directory-pointer table																							Ign.	XU[4]	Ign.	A	Reserved						X[5]	W	R	**PML4E: present[6]**
SVE[7]	Ignored																																													**0**	**0**	**0**	**PML4E: not present**
SVE	Ign.		SSS[8]	Ign.	PW[9]	VGP[10]	Ignored						Rsvd.	Physical address of 1GB page																							Ign.	XU	D	A	**1**	IPAT	EPT MT				X	W	R	**PDPTE: 1GB page**
Ignored													Rsvd.	Address of EPT page directory																							Ign.	XU	Ign.	A	**0**	Rsvd.					X	W	R	**PDPTE: page directory**
SVE	Ignored																																													**0**	**0**	**0**	**PDTPE: not present**
SVE	Ign.		SSS	Ign.	PW	VGP	Ignored						Rsvd.	Physical address of 2MB page																							Ign.	XU	D	A	**1**	IPAT	EPT MT				X	W	R	**PDE: 2MB page**
Ignored													Rsvd.	Address of EPT page table																							Ign.	XU	Ign.	A	**0**	Rsvd.					X	W	R	**PDE: page table**
SVE	Ignored																																													**0**	**0**	**0**	**PDE: not present**
SVE	Ign.		SPP[11]	SSS	Ign.	PW	VGP	Ignored					Rsvd.	Physical address of 4KB page																							Ign.	XU	D	A	Ign	IPAT	EPT MT				X	W	R	**PTE: 4KB page**
SVE	Ignored																																													**0**	**0**	**0**	**PTE: not present**

Figure 28-1. Formats of EPTP and EPT Paging-Structure Entries

NOTES:

1. M is an abbreviation for MAXPHYADDR.
2. Supervisor shadow-stack control.
3. See Section 24.6.11 for details of the EPTP.
4. Execute access for user-mode linear addresses. If the "mode-based execute control for EPT" VM-execution control is 0, this bit is ignored.
5. Execute access. If the "mode-based execute control for EPT" VM-execution control is 1, this bit controls execute access for supervisor-mode linear addresses.
6. 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. This table does not comprehend that fact.
7. Suppress #VE. If the "EPT-violation #VE" VM-execution control is 0, this bit is ignored.
8. Supervisor shadow-stack page. If bit 7 of the EPTP is 0, this bit is ignored.

9. Paging-write access. If the "EPT paging-write control" VM-execution control is 0, this bit is ignored.
10. Verify guest paging. If the "guest-paging verification" VM-execution control is 0, this bit is ignored.
11. Sub-page write permissions. If the "sub-page write permissions for EPT" VM-execution control is 0, this bit is ignored.

- The access is a data write and, for any byte to be written, bit 1 (write access) was clear in any of the EPT paging-structure entries used to translate the guest-physical address of the byte. Writes by the logical processor to guest paging structures to update accessed and dirty flags are considered to be data writes.

  If bit 6 of the EPT pointer (EPTP) is 1 (enabling accessed and dirty flags for EPT), processor accesses to guest paging-structure entries are treated as writes with regard to EPT violations. Thus, if bit 1 is clear in any of the EPT paging-structure entries used to translate the guest-physical address of a guest paging-structure entry, an attempt to use that entry to translate a linear address causes an EPT violation.

  (This does not apply to loads of the PDPTE registers by the MOV to CR instruction for PAE paging; see Section 4.4.1. Those loads of guest PDPTEs are treated as reads and do not cause EPT violations due to a guest-physical address not being writable.)

  Processor writes to guest paging structures to update accessed and dirty flags are called **paging writes**. (When accessed and dirty flags for EPT are enabled all processor accesses to guest paging structures are considered paging writes.) If the "EPT paging-write control" VM-execution control is 1, clearing the write-access bit in the EPT paging-structure entry that maps a page does not prevent a paging write if bit 58 (paging-write access) in that entry is 1. (EPT paging-structure entries that reference other EPT paging structures do not use bit 58, and it remains the case that they must set the write-access bit for paging writes to be allowed.)

  If the "sub-page write permissions for EPT" VM-execution control is 1, data writes to a guest-physical address that would cause an EPT violation (as indicated above) do not do so in certain situations. If the guest-physical address is mapped using a 4-KByte page and bit 61 (sub-page write permissions) of the EPT PTE used to map the page is 1, writes to certain 128-byte sub-pages may be allowed. See Section 28.3.4 for details.
- The access is an instruction fetch and the EPT paging structures prevent execute access to any of the bytes being fetched. Whether this occurs depends upon the setting of the "mode-based execute control for EPT" VM-execution control:
  - If the control is 0, an instruction fetch from a byte is prevented if bit 2 (execute access) was clear in any of the EPT paging-structure entries used to translate the guest-physical address of the byte.
  - If the control is 1, an instruction fetch from a byte is prevented in either of the following cases:
    - Paging maps the linear address of the byte as a supervisor-mode address and bit 2 (execute access for supervisor-mode linear addresses) was clear in any of the EPT paging-structure entries used to translate the guest-physical address of the byte.

      Paging maps a linear address as a supervisor-mode address if the U/S flag (bit 2) is 0 in at least one of the paging-structure entries controlling the translation of the linear address.
    - Paging maps the linear address of the byte as a user-mode address and bit 10 (execute access for user-mode linear addresses) was clear in any of the EPT paging-structure entries used to translate the guest-physical address of the byte.

      Paging maps a linear address as a user-mode address if the U/S flag is 1 in all of the paging-structure entries controlling the translation of the linear address. If paging is disabled (CR0.PG = 0), every linear address is a user-mode address.
- If supervisor shadow-stack control is enabled (by setting bit 7 of EPTP), the access is a supervisor shadow-stack access, and the EPT paging-structure entries used to translate the guest-physical address of the access disallow supervisor shadow-stack accesses. Such an access is disallowed if any of the following hold:
  - Bit 0 (read access) is clear in any EPT paging-structure entry used to translate the guest-physical address of the access.
  - Bit 1 (write access) is clear in any EPT paging-structure entry that references an EPT paging structure in the translation of the guest-physical address. (Clearing bit 1 in the EPT paging-structure entry that maps the page of the guest-physical address does not disallow shadow-stack reads and writes.)
  - Bit 60 (supervisor-shadow stack access) is clear in the EPT paging-structure entry that maps the page of the guest-physical address.

Supervisor shadow-stack control and the supervisor-shadow stack access bits in EPT paging-structure entries do not affect other accesses (including user shadow-stack accesses).

- If the "guest-paging verification" and "EPT paging-write control" VM-execution controls are both 1, **guest-paging verification** is performed for certain accesses related to translation of a linear address.

  Specifically, guest-paging verification may apply to an access using a guest-physical address to the guest paging-structure entry that maps a page for the linear address (see Chapter 4, "Paging"). It applies if bit 57 (verify guest paging) is set in the EPT paging-structure entry that maps a page for that guest-physical address.

  When guest-paging verification occurs, there is an EPT violation if bit 58 (paging-write access) is clear in the EPT paging-structure entry that was used to map a page for the guest-physical address of any guest paging-structure entry that was used during translation of the original linear address.[1]

  See Section 28.3.3.3 for details regarding the prioritization of such EPT violations relative to any page faults that may occur due to the original reference to the linear address being translated.

28.3.3.3 Prioritization of EPT Misconfigurations and EPT Violations

The translation of a linear address to a physical address requires one or more translations of guest-physical addresses using EPT (see Section 28.3.1). This section specifies the relative priority of EPT-induced VM exits with respect to each other and to other events that may be encountered when accessing memory using a linear address.

For an access to a guest-physical address, determination of whether an EPT misconfiguration or an EPT violation occurs is based on an iterative process:[2]

1. An EPT paging-structure entry is read (initially, this is an EPT PML4 entry):
   a. If the entry is not present (see Section 28.3.2), an EPT violation occurs.
   b. If the entry is present but its contents are not configured properly (see Section 28.3.3.1), an EPT misconfiguration occurs.
   c. If the entry is present and its contents are configured properly, operation depends on whether the entry references another EPT paging structure (whether it is an EPT PDE with bit 7 set to 1 or an EPT PTE):
      i) If the entry does reference another EPT paging structure, an entry from that structure is accessed; step 1 is executed for that other entry.
      ii) Otherwise, the entry is used to produce the ultimate physical address (the translation of the original guest-physical address); step 2 is executed.
2. Once the ultimate physical address is determined, the privileges determined by the EPT paging-structure entries are evaluated:
   a. If the access to the guest-physical address is not allowed by these privileges (see Section 28.3.3.2), an EPT violation occurs.
   b. If the access to the guest-physical address is allowed by these privileges, memory is accessed using the ultimate physical address.

If CR0.PG = 1, the translation of a linear address is also an iterative process, with the processor first accessing an entry in the guest paging structure referenced by the guest-physical address in CR3,[3] then accessing an entry in another guest paging structure referenced by the guest-physical address in the first guest paging-structure entry, etc. Each guest-physical address is itself translated using EPT and may cause an EPT-induced VM exit. The following items detail how page faults and EPT-induced VM exits are recognized during this iterative process:

1. An attempt is made to access a guest paging-structure entry with a guest-physical address (initially, the address in CR3.

---

1. When there is a restart of HLAT paging (see Section 4.5), the guest paging-structure entries used by HLAT paging (prior to the restart) are not relevant to guest-paging verification.
2. This is a simplification of the more detailed description given in Section 28.3.2.
3. If PAE paging is in use, the processor instead uses the guest-physical address in the appropriate PDPTE register. If HLAT paging is enabled, the processor instead uses the HLATP VMCS field. For simplicity, the remainder of this section refers only to CR3.

a. If the access fails because of an EPT misconfiguration or an EPT violation (see above), an EPT-induced VM exit occurs.

b. If the access does not cause an EPT-induced VM exit, the translation continues. If guest-paging verification is enabled (see Section 28.3.3.2), the processor notes whether the EPT paging-structure entry used to map a page for the guest-physical address set bit 58 (paging write). Then, bit 0 (the present flag) of the guest paging-structure entry is consulted:

   i) If the present flag is 0 or any reserved bit is set, a page fault occurs.

   ii) If the present flag is 1, no reserved bit is set, operation depends on whether the entry references another guest paging structure (whether it is a guest PDE with PS = 1 or a guest PTE):

   - If the entry does reference another guest paging structure, an entry from that structure is accessed; step 1 is executed for that other entry.
   - Otherwise, the entry is used to produce the ultimate guest-physical address (the translation of the original linear address); step 2 is executed.

2. Once the ultimate guest-physical address is determined, the privileges determined by the guest paging-structure entries are evaluated:

   a. If the access to the linear address is not allowed by these privileges (e.g., it was a write to a read-only page), a page fault occurs.

   b. If the access to the linear address is allowed by these privileges, an attempt is made to access memory at the ultimate guest-physical address:

      i) If the access fails because of an EPT misconfiguration or an EPT violation (see above), an EPT-induced VM exit occurs. It is at this point that guest-paging verification occurs (if enabled), using the paging-write bits that were noted at occurrences of step 1b above (see Section 28.3.3.2 for details of guest-paging verification).

      ii) If the access does not cause an EPT-induced VM exit, memory is accessed using the ultimate physical address (the translation, using EPT, of the ultimate guest-physical address).

If CR0.PG = 0, a linear address is treated as a guest-physical address and is translated using EPT (see above). This process, if it completes without an EPT violation or EPT misconfiguration, produces a physical address and determines the privileges allowed by the EPT paging-structure entries. If these privileges do not allow the access to the physical address (see Section 28.3.3.2), an EPT violation occurs. Otherwise, memory is accessed using the physical address.

28.3.4 Sub-Page Write Permissions

Section 28.3.3.2 explained how EPT enforces the access rights for guest-physical addresses using EPT violations. Since these access rights are determined using the EPT paging-structure entries that are used to translate a guest-physical address, their granularity is limited to that which is used to map pages (1-GByte, 2-MByte, or 4-KByte).

The **sub-page write-permission** feature allows the control of write accesses to guest-physical addresses to be controlled at finer granularity. Sub-page write permissions allow write accesses to be controlled at the granularity of naturally aligned 128-byte **sub-pages**. Specifically, the feature allows writes to selected sub-pages of 4-KByte page that would otherwise not be writable.

Sub-page write permissions are enabled setting the "sub-page write permissions for EPT" VM-execution control to 1. The remainder of this section describes changes to processor operation with this control setting.

Section 28.3.4.1 identifies the data accesses that are eligible for sub-page write permissions. Section 28.3.4.2 explains how the processor determines whether to allow such an access.

28.3.4.1 Write Accesses That Are Eligible for Sub-Page Write Permissions

A guest-physical address is eligible for sub-page write permissions if writes to it would be disallowed following Section 28.3.3.2: bit 1 (write access) is clear in any of the EPT paging-structure entries used to translate the guest-

physical address. Guest-physical addresses to which writes would be disallowed for other reasons (e.g., the translation encounters an EPT paging-structure entry that is not present) are not eligible for sub-page write permissions.

In addition, a guest-physical address is eligible for sub-page write permissions only if it is mapped using a 4-KByte page and bit 61 (sub-page write permissions) of the EPT PTE used to map the page is 1. (Guest-physical addresses mapped with larger pages are not eligible for sub-page write permissions.)

For some memory accesses, the processor ignores bit 61 in an EPT PTE used to map a 4-KByte page and does not apply sub-page write permissions to the access. (In such a case, the access causes an EPT violation when indicated by the conditions given in Section 28.3.3.2.) Sub-page write permissions never apply to the following accesses:

- A write access performed within a transactional region.
- A write access by an enclave to an address within the enclave's ELRANGE. (Sub-page write permissions may apply to write accesses by an enclaves to addresses outside its ELRANGE.)
- A write access to the enclave page cache (EPC) by an Intel SGX instruction.
- A write access to a guest paging structure to update an accessed or dirty flag.
- Processor accesses to guest paging-structure entries when accessed and dirty flags for EPT are enabled (such accesses are treated as writes with regard to EPT violations.

There are additional accesses to which sub-page write permissions might not be applied (behavior is model-specific). The following items enumerate examples:

- A write access that crosses two 4-KByte pages. In this case, sub-page permissions may be applied to neither or to only one of the pages. (There is no write to either page unless the write is allowed to both pages.)
- A write access by an instruction that performs multiple write accesses (sub-page write permissions are intended principally for basic instructions such as AND, MOV, OR, TEST, XCHG, and XOR).

If a guest-physical address is eligible for sub-page write permissions, the processor determines whether to allow write to the address using the process described in Section 28.3.4.2.

If a guest-physical address is eligible for sub-page write permissions and that address translates to an address on the APIC-access page (see Section 29.4), the processor may treat a write access to the address as if the "virtualize APIC accesses" VM-execution control were 0. 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.

28.3.4.2 Determining an Access's Sub-Page Write Permission

Sub-page write permissions control write accesses individually to each of the 32 128-byte sub-pages of a 4-KByte page. Bits 11:7 of guest-physical address identify the sub-page.

For each guest-physical address eligible for sub-page write permissions, there is a 64-bit **sub-page permission vector** (**SPP vector**). All addresses on a 4-KByte page use the same SPP vector. If an address's sub-page number (bits 11:7 of the address) is S, writes to address are allowed if and only if bit 2S of the sub-page permission is set to 1. (The bits at odd positions in a SPP vector are not used and must be zero.)

Each page's SPP vector is located in memory. For a write to a guest-physical address eligible for sub-page write permissions, the processor uses the following process to locate the address's SPP vector:

1. The SPPTP (sub-page-permission-table pointer) VM-execution control field contains the physical address of the 4-KByte root SPP table (SSPL4 table). Bits 47:39 of the guest-physical address identify a 64-bit entry in that table, called an SPPL4E.
2. A 4-KByte SPPL3 table is located at the physical address in the selected SPPL4E. Bits 38:30 of the guest-physical address identify a 64-bit entry in that table, called an SPPL3E.
3. A 4-KByte SPPL2 table is located at the physical address in the selected SPPL3E. Bits 29:21 of the guest-physical address identify a 64-bit entry in that table, called an SPPL2E.
4. A 4-KByte SPP-vector table (SSPL1 table) is located at the physical address in the selected SPPL2E. Bits 20:12 of the guest-physical address identify the 64-bit SPP vector for the address. As noted earlier, bit 2S of the sub-page permission vector determines whether the address may be written, where S is the value of address bits 11:7.

(The memory type used to access these tables is reported in bits 53:50 of the IA32_VMX_BASIC MSR. See Appendix A.1.)

A write access to multiple 128-byte sub-pages on a single 4-KByte page is allowed only if the indicated bit in the page's SPP vector for each of those sub-pages. The following items apply to cases in which an access writes to two 4-KByte pages:

- If a write to either page would be disallowed according to Section 28.2.3.2, the access might be disallowed even if the guest-physical address of that page is eligible for sub-page write permissions. (This behavior is model-specific.)
- The access is allowed only if, for each page, either (1) a write to the page would be allowed following Section 28.2.3.2; or (2) both (a) the guest-physical address of that page is eligible for sub-page write permissions; and (b) the page's sub-page vector allows the write (as described above).

Bit 0 of each entry (SPPL4E, SPPL3E, or SPPL2E) is the entry's **valid** bit. If the process above accesses an entry in which this bit is 0, the process stops and the logical processor incurs an **SPP miss**.

In each entry (SPPL4E, SPPL4E, or SPPL2E), bits 11:1 are reserved, as are bits 63:N, where N is the processor's physical-address width. If the process above accesses an entry in which the valid bit is 1 and in which some reserved bit is set, the process stops and the logical processor incurs an **SPP misconfiguration**. Bits in an SPP vector in odd positions are also reserved; an SPP misconfiguration occurs also any of those bits are set in the final SPP vector.

SPP misses and SPP misconfigurations are called **SPP-related events** and cause VM exits.

28.3.5 Accessed and Dirty Flags for EPT

The Intel 64 architecture supports **accessed and dirty flags** in ordinary paging-structure entries (see Section 4.8). Some processors also support corresponding flags in EPT paging-structure entries. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.

Software can enable accessed and dirty flags for EPT using bit 6 of the extended-page-table pointer (EPTP), a VM-execution control field (see Table 24-9 in Section 24.6.11). If this bit is 1, the processor will set the accessed and dirty flags for EPT as described below. In addition, setting this flag causes processor accesses to guest paging-structure entries to be treated as writes (see below and Section 28.3.3.2).

For any EPT paging-structure entry that is used during guest-physical-address translation, bit 8 is the accessed flag. For a EPT paging-structure entry that maps a page (as opposed to referencing another EPT paging structure), bit 9 is the dirty flag.

Whenever the processor uses an EPT paging-structure entry as part of guest-physical-address translation, it sets the accessed flag in that entry (if it is not already set).

Whenever there is a write to a guest-physical address, the processor sets the dirty flag (if it is not already set) in the EPT paging-structure entry that identifies the final physical address for the guest-physical address (either an EPT PTE or an EPT paging-structure entry in which bit 7 is 1).

When accessed and dirty flags for EPT are enabled, processor accesses to guest paging-structure entries are treated as writes (see Section 28.3.3.2). Thus, such an access will cause the processor to set the dirty flag in the EPT paging-structure entry that identifies the final physical address of the guest paging-structure entry.

(This does not apply to loads of the PDPTE registers for PAE paging by the MOV to CR instruction; see Section 4.4.1. Those loads of guest PDPTEs are treated as reads and do not cause the processor to set the dirty flag in any EPT paging-structure entry.)

These flags are "sticky," meaning that, once set, the processor does not clear them; only software can clear them.

A processor may cache information from the EPT paging-structure entries in TLBs and paging-structure caches (see Section 28.4). This fact implies that, if software changes an accessed flag or a dirty flag from 1 to 0, the processor might not set the corresponding bit in memory on a subsequent access using an affected guest-physical address.

28.3.6 Page-Modification Logging

When accessed and dirty flags for EPT are enabled, software can track writes to guest-physical addresses using a feature called **page-modification logging**.

Software can enable page-modification logging by setting the "enable PML" VM-execution control (see Table 24-7 in Section 24.6.2). When this control is 1, the processor adds entries to the **page-modification log** as described below. The page-modification log is a 4-KByte region of memory located at the physical address in the PML address VM-execution control field. The page-modification log consists of 512 64-bit entries; the PML index VM-execution control field indicates the next entry to use.

Before allowing a guest-physical access, the processor may determine that it first needs to set an accessed or dirty flag for EPT (see Section 28.3.5). When this happens, the processor examines the PML index. If the PML index is not in the range 0–511, there is a **page-modification log-full event** and a VM exit occurs. In this case, the accessed or dirty flag is not set, and the guest-physical access that triggered the event does not occur.

If instead the PML index is in the range 0–511, the processor proceeds to update accessed or dirty flags for EPT as described in Section 28.3.5. If the processor updated a dirty flag for EPT (changing it from 0 to 1), it then operates as follows:

1. The guest-physical address of the access is written to the page-modification log. Specifically, the guest-physical address is written to physical address determined by adding 8 times the PML index to the PML address. Bits 11:0 of the value written are always 0 (the guest-physical address written is thus 4-KByte aligned).
2. The PML index is decremented by 1 (this may cause the value to transition from 0 to FFFFH).

Because the processor decrements the PML index with each log entry, the value may transition from 0 to FFFFH. At that point, no further logging will occur, as the processor will determine that the PML index is not in the range 0–511 and will generate a page-modification log-full event (see above).

28.3.7 EPT and Memory Typing

This section specifies how a logical processor determines the memory type use for a memory access while EPT is in use. (See Chapter 11, "Memory Cache Control" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* for details of memory typing in the Intel 64 architecture.) Section 28.3.7.1 explains how the memory type is determined for accesses to the EPT paging structures. Section 28.3.7.2 explains how the memory type is determined for an access using a guest-physical address that is translated using EPT.

28.3.7.1 Memory Type Used for Accessing EPT Paging Structures

This section explains how the memory type is determined for accesses to the EPT paging structures. The determination is based first on the value of bit 30 (cache disable—CD) in control register CR0:

- If CR0.CD = 0, the memory type used for any such reference is the EPT paging-structure memory type, which is specified in bits 2:0 of the extended-page-table pointer (EPTP), a VM-execution control field (see Section 24.6.11). A value of 0 indicates the uncacheable type (UC), while a value of 6 indicates the write-back type (WB). Other values are reserved.
- If CR0.CD = 1, the memory type used for any such reference is uncacheable (UC).

The MTRRs have no effect on the memory type used for an access to an EPT paging structure.

28.3.7.2 Memory Type Used for Translated Guest-Physical Addresses

The **effective memory type** of a memory access using a guest-physical address (an access that is translated using EPT) is the memory type that is used to access memory. The effective memory type is based on the value of bit 30 (cache disable—CD) in control register CR0; the **last** EPT paging-structure entry used to translate the guest-physical address (either an EPT PDE with bit 7 set to 1 or an EPT PTE); and the PAT memory type (see below):

- The **PAT memory type** depends on the value of CR0.PG:
  - If CR0.PG = 0, the PAT memory type is WB (writeback).[1]

  - If CR0.PG = 1, the PAT memory type is the memory type selected from the IA32_PAT MSR as specified in Section 11.12.3, "Selecting a Memory Type from the PAT".[1]
- The **EPT memory type** is specified in bits 5:3 of the last EPT paging-structure entry: 0 = UC; 1 = WC; 4 = WT; 5 = WP; and 6 = WB. Other values are reserved and cause EPT misconfigurations (see Section 28.3.3).
- If CR0.CD = 0, the effective memory type depends upon the value of bit 6 of the last EPT paging-structure entry:
  - If the value is 0, the effective memory type is the combination of the EPT memory type and the PAT memory type specified in Table 11-7 in Section 11.5.2.2, using the EPT memory type in place of the MTRR memory type.
  - If the value is 1, the memory type used for the access is the EPT memory type. The PAT memory type is ignored.
- If CR0.CD = 1, the effective memory type is UC.

The MTRRs have no effect on the memory type used for an access to a guest-physical address.

28.4 CACHING TRANSLATION INFORMATION

Processors supporting Intel® 64 and IA-32 architectures may accelerate the address-translation process by caching on the processor data from the structures in memory that control that process. Such caching is discussed in Section 4.10, "Caching Translation Information" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.* The current section describes how this caching interacts with the VMX architecture.

The VPID and EPT features of the architecture for VMX operation augment this caching architecture. EPT defines the guest-physical address space and defines translations to that address space (from the linear-address space) and from that address space (to the physical-address space). Both features control the ways in which a logical processor may create and use information cached from the paging structures.

Section 28.4.1 describes the different kinds of information that may be cached. Section 28.4.2 specifies when such information may be cached and how it may be used. Section 28.4.3 details how software can invalidate cached information.

28.4.1 Information That May Be Cached

Section 4.10, "Caching Translation Information" in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* identifies two kinds of translation-related information that may be cached by a logical processor: **translations**, which are mappings from linear page numbers to physical page frames, and **paging-structure caches**, which map the upper bits of a linear page number to information from the paging-structure entries used to translate linear addresses matching those upper bits.

The same kinds of information may be cached when VPIDs and EPT are in use. A logical processor may cache and use such information based on its function. Information with different functionality is identified as follows:

- **Linear mappings**.[2] There are two kinds:
  - Linear translations. Each of these is a mapping from a linear page number to the physical page frame to which it translates, along with information about access privileges and memory typing.

---

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG can be 0 in VMX non-root operation only if the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

1. Table 11-11 in Section 11.12.3, "Selecting a Memory Type from the PAT" illustrates how the PAT memory type is selected based on the values of the PAT, PCD, and PWT bits in a page-table entry (or page-directory entry with PS = 1). For accesses to a guest paging-structure entry X, the PAT memory type is selected from the table by using a value of 0 for the PAT bit with the values of PCD and PWT from the paging-structure entry Y that references X (or from CR3 if X is in the root paging structure). With PAE paging, the PAT memory type for accesses to the PDPTEs is WB.

2. Earlier versions of this manual used the term "VPID-tagged" to identify linear mappings.

— Linear paging-structure-cache entries. Each of these is a mapping from the upper portion of a linear address to the physical address of the paging structure used to translate the corresponding region of the linear-address space, along with information about access privileges. For example, bits 47:39 of a linear address would map to the address of the relevant page-directory-pointer table.

Linear mappings do not contain information from any EPT paging structure.

- **Guest-physical mappings**.[1] There are two kinds:
  - — Guest-physical translations. Each of these is a mapping from a guest-physical page number to the physical page frame to which it translates, along with information about access privileges and memory typing.
  - — Guest-physical paging-structure-cache entries. Each of these is a mapping from the upper portion of a guest-physical address to the physical address of the EPT paging structure used to translate the corresponding region of the guest-physical address space, along with information about access privileges.

  The information in guest-physical mappings about access privileges and memory typing is derived from EPT paging structures.
- **Combined mappings**.[2] There are two kinds:
  - — Combined translations. Each of these is a mapping from a linear page number to the physical page frame to which it translates, along with information about access privileges and memory typing.
  - — Combined paging-structure-cache entries. Each of these is a mapping from the upper portion of a linear address to the physical address of the paging structure used to translate the corresponding region of the linear-address space, along with information about access privileges.

  The information in combined mappings about access privileges and memory typing is derived from both guest paging structures and EPT paging structures.

Guest-physical mappings and combined mappings may also include SPP vectors and information about the data structures used to locate SPP vectors (see Section 28.3.4.2).

28.4.2 Creating and Using Cached Translation Information

The following items detail the creation of the mappings described in the previous section:[3]

- The following items describe the creation of mappings while EPT is not in use (including execution outside VMX non-root operation):
  - — Linear mappings may be created. They are derived from the paging structures referenced (directly or indirectly) by the current value of CR3 and are associated with the current VPID and the current PCID.
  - — No linear mappings are created with information derived from paging-structure entries that are not present (bit 0 is 0) or that set reserved bits. For example, if a PTE is not present, no linear mapping are created for any linear page number whose translation would use that PTE.
  - — No guest-physical or combined mappings are created while EPT is not in use.
- The following items describe the creation of mappings while EPT is in use:
  - — Guest-physical mappings may be created. They are derived from the EPT paging structures referenced (directly or indirectly) by bits 51:12 of the current EPTP. These 40 bits contain the address of the EPT-PML4-table. (the notation **EP4TA** refers to those 40 bits). Newly created guest-physical mappings are associated with the current EP4TA.
  - — Combined mappings may be created. They are derived from the EPT paging structures referenced (directly or indirectly) by the current EP4TA. If CR0.PG = 1, they are also derived from the paging structures referenced (directly or indirectly) by the current value of CR3. They are associated with the current VPID,

1. Earlier versions of this manual used the term "EPTP-tagged" to identify guest-physical mappings.
2. Earlier versions of this manual used the term "dual-tagged" to identify combined mappings.
3. This section associated cached information with the current VPID and PCID. If PCIDs are not supported or are not being used (e.g., because CR4.PCIDE = 0), all the information is implicitly associated with PCID 000H; see Section 4.10.1, "Process-Context Identifiers (PCIDs)," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

the current PCID, and the current EP4TA.[1] No combined paging-structure-cache entries are created if CR0.PG = 0.[2]

— No guest-physical mappings or combined mappings are created with information derived from EPT paging-structure entries that are not present (see Section 28.3.2) or that are misconfigured (see Section 28.3.3.1).

— No combined mappings are created with information derived from guest paging-structure entries that are not present or that set reserved bits.

— No linear mappings are created while EPT is in use.

The following items detail the use of the various mappings:

- If EPT is not in use (e.g., when outside VMX non-root operation), a logical processor may use cached mappings as follows:
  - For accesses using linear addresses, it may use linear mappings associated with the current VPID and the current PCID. It may also use global TLB entries (linear mappings) associated with the current VPID and any PCID.
  - No guest-physical or combined mappings are used while EPT is not in use.
- If EPT is in use, a logical processor may use cached mappings as follows:
  - For accesses using linear addresses, it may use combined mappings associated with the current VPID, the current PCID, and the current EP4TA. It may also use global TLB entries (combined mappings) associated with the current VPID, the current EP4TA, and any PCID.
  - For accesses using guest-physical addresses, it may use guest-physical mappings associated with the current EP4TA.
  - No linear mappings are used while EPT is in use.

28.4.3 Invalidating Cached Translation Information

Software modifications of paging structures (including EPT paging structures and the data structures used to locate SPP vectors) may result in inconsistencies between those structures and the mappings cached by a logical processor. Certain operations invalidate information cached by a logical processor and can be used to eliminate such inconsistencies.

28.4.3.1 Operations that Invalidate Cached Mappings

The following operations invalidate cached mappings as indicated:

- Operations that architecturally invalidate entries in the TLBs or paging-structure caches independent of VMX operation (e.g., the INVLPG and INVPCID instructions) invalidate linear mappings and combined mappings.[3] They are required to do so only for the current VPID (but, for combined mappings, all EP4TAs). Linear mappings for the current VPID are invalidated even if EPT is in use.[4] Combined mappings for the current VPID are invalidated even if EPT is not in use.[5]

---

1. At any given time, a logical processor may be caching combined mappings for a VPID and a PCID that are associated with different EP4TAs. Similarly, it may be caching combined mappings for an EP4TA that are associated with different VPIDs and PCIDs.
2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG can be 0 in VMX non-root operation only if the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
3. See Section 4.10.4, "Invalidation of TLBs and Paging-Structure Caches," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* for an enumeration of operations that architecturally invalidate entries in the TLBs and paging-structure caches independent of VMX operation.
4. While no linear mappings are created while EPT is in use, a logical processor may retain, while EPT is in use, linear mappings (for the same VPID as the current one) there were created earlier, when EPT was not in use.
5. While no combined mappings are created while EPT is not in use, a logical processor may retain, while EPT is in not use, combined mappings (for the same VPID as the current one) there were created earlier, when EPT was in use.

- An EPT violation invalidates any guest-physical mappings (associated with the current EP4TA) that would be used to translate the guest-physical address that caused the EPT violation. If that guest-physical address was the translation of a linear address, the EPT violation also invalidates any combined mappings for that linear address associated with the current PCID, the current VPID and the current EP4TA.
- If the "enable VPID" VM-execution control is 0, VM entries and VM exits invalidate linear mappings and combined mappings associated with VPID 0000H (for all PCIDs). Combined mappings for VPID 0000H are invalidated for all EP4TAs.
- Execution of the INVVPID instruction invalidates linear mappings and combined mappings. Invalidation is based on instruction operands, called the INVVPID type and the INVVPID descriptor. Four INVVPID types are currently defined:
  - **Individual-address.** If the INVVPID type is 0, the logical processor invalidates linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor and that would be used to translate the linear address specified in of the INVVPID descriptor. Linear mappings and combined mappings for that VPID and linear address are invalidated for all PCIDs and, for combined mappings, all EP4TAs. (The instruction may also invalidate mappings associated with other VPIDs and for other linear addresses.)
  - **Single-context.** If the INVVPID type is 1, the logical processor invalidates all linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor. Linear mappings and combined mappings for that VPID are invalidated for all PCIDs and, for combined mappings, all EP4TAs. (The instruction may also invalidate mappings associated with other VPIDs.)
  - **All-context.** If the INVVPID type is 2, the logical processor invalidates linear mappings and combined mappings associated with all VPIDs except VPID 0000H and with all PCIDs. (The instruction may also invalidate linear mappings with VPID 0000H.) Combined mappings are invalidated for all EP4TAs.
  - **Single-context-retaining-globals.** If the INVVPID type is 3, the logical processor invalidates linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor. Linear mappings and combined mappings for that VPID are invalidated for all PCIDs and, for combined mappings, all EP4TAs. The logical processor is **not** required to invalidate information that was used for **global** translations (although it may do so). See Section 4.10, "Caching Translation Information" for details regarding global translations. (The instruction may also invalidate mappings associated with other VPIDs.)

  See Chapter 30 for details of the INVVPID instruction. See Section 28.4.3.3 for guidelines regarding use of this instruction.
- Execution of the INVEPT instruction invalidates guest-physical mappings and combined mappings. Invalidation is based on instruction operands, called the INVEPT type and the INVEPT descriptor. Two INVEPT types are currently defined:
  - **Single-context.** If the INVEPT type is 1, the logical processor invalidates all guest-physical mappings and combined mappings associated with the EP4TA specified in the INVEPT descriptor. Combined mappings for that EP4TA are invalidated for all VPIDs and all PCIDs. (The instruction may invalidate mappings associated with other EP4TAs.)
  - **All-context.** If the INVEPT type is 2, the logical processor invalidates guest-physical mappings and combined mappings associated with all EP4TAs (and, for combined mappings, for all VPIDs and PCIDs).

  See Chapter 30 for details of the INVEPT instruction. See Section 28.4.3.4 for guidelines regarding use of this instruction.
- A power-up or a reset invalidates all linear mappings, guest-physical mappings, and combined mappings.

28.4.3.2 Operations that Need Not Invalidate Cached Mappings

The following items detail cases of operations that are not required to invalidate certain cached mappings:

- Operations that architecturally invalidate entries in the TLBs or paging-structure caches independent of VMX operation are not required to invalidate any guest-physical mappings.
- The INVVPID instruction is not required to invalidate any guest-physical mappings.
- The INVEPT instruction is not required to invalidate any linear mappings.

- VMX transitions are not required to invalidate any guest-physical mappings. If the "enable VPID" VM-execution control is 1, VMX transitions are not required to invalidate any linear mappings or combined mappings.
- The VMXOFF and VMXON instructions are not required to invalidate any linear mappings, guest-physical mappings, or combined mappings.

A logical processor may invalidate any cached mappings at any time. For this reason, the operations identified above may invalidate the indicated mappings despite the fact that doing so is not required.

28.4.3.3 Guidelines for Use of the INVVPID Instruction

The need for VMM software to use the INVVPID instruction depends on how that software is virtualizing memory.

If EPT is not in use, it is likely that the VMM is virtualizing the guest paging structures. Such a VMM may configure the VMCS so that all or some of the operations that invalidate entries the TLBs and the paging-structure caches (e.g., the INVLPG instruction) cause VM exits. If VMM software is emulating these operations, it may be necessary to use the INVVPID instruction to ensure that the logical processor's TLBs and the paging-structure caches are appropriately invalidated.

Requirements of when software should use the INVVPID instruction depend on the specific algorithm being used for page-table virtualization. The following items provide guidelines for software developers:

- Emulation of the INVLPG instruction may require execution of the INVVPID instruction as follows:
  - The INVVPID type is individual-address (0).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.
  - The linear address in the INVVPID descriptor is that of the operand of the INVLPG instruction being emulated.
- Some instructions invalidate all entries in the TLBs and paging-structure caches—except for global translations. An example is the MOV to CR3 instruction. (See Section 4.10, "Caching Translation Information" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* for details regarding global translations.) Emulation of such an instruction may require execution of the INVVPID instruction as follows:
  - The INVVPID type is single-context-retaining-globals (3).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.
- Some instructions invalidate all entries in the TLBs and paging-structure caches—including for global translations. An example is the MOV to CR4 instruction if the value of value of bit 4 (page global enable—PGE) is changing. Emulation of such an instruction may require execution of the INVVPID instruction as follows:
  - The INVVPID type is single-context (1).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.

If EPT is not in use, the logical processor associates all mappings it creates with the current VPID, and it will use such mappings to translate linear addresses. For that reason, a VMM should not use the same VPID for different non-EPT guests that use different page tables. Doing so may result in one guest using translations that pertain to the other.

If EPT is in use, the instructions enumerated above might not be configured to cause VM exits and the VMM might not be emulating them. In that case, executions of the instructions by guest software properly invalidate the required entries in the TLBs and paging-structure caches (see Section 28.4.3.1); execution of the INVVPID instruction is not required.

If EPT is in use, the logical processor associates all mappings it creates with the value of bits 51:12 of current EPTP. If a VMM uses different EPTP values for different guests, it may use the same VPID for those guests. Doing so cannot result in one guest using translations that pertain to the other.

The following guidelines apply more generally and are appropriate even if EPT is in use:

- As detailed in Section 29.4.5, an access to the APIC-access page might not cause an APIC-access VM exit if software does not properly invalidate information that may be cached from the paging structures. If, at one

time, the current VPID on a logical processor was a non-zero value X, it is recommended that software use the INVVPID instruction with the "single-context" INVVPID type and with VPID X in the INVVPID descriptor before a VM entry on the same logical processor that establishes VPID X and either (a) the "virtualize APIC accesses" VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.

- Software can use the INVVPID instruction with the "all-context" INVVPID type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from paging structures between separate uses of VMX operation.

28.4.3.4 Guidelines for Use of the INVEPT Instruction

The following items provide guidelines for use of the INVEPT instruction to invalidate information cached from the EPT paging structures.

- Software should use the INVEPT instruction with the "single-context" INVEPT type after making any of the following changes to an EPT paging-structure entry (the INVEPT descriptor should contain an EPTP value that references — directly or indirectly — the modified EPT paging structure):
  - Changing any of the privilege bits 2:0 from 1 to 0.[1]
  - Changing the physical address in bits 51:12.
  - Clearing bit 8 (the accessed flag) if accessed and dirty flags for EPT will be enabled.
  - For an EPT PDPTE or an EPT PDE, changing bit 7 (which determines whether the entry maps a page).
  - For the **last** EPT paging-structure entry used to translate a guest-physical address (an EPT PDPTE with bit 7 set to 1, an EPT PDE with bit 7 set to 1, or an EPT PTE), changing either bits 5:3 or bit 6. (These bits determine the effective memory type of accesses using that EPT paging-structure entry; see Section 28.3.7.)
  - For the **last** EPT paging-structure entry used to translate a guest-physical address (an EPT PDPTE with bit 7 set to 1, an EPT PDE with bit 7 set to 1, or an EPT PTE), clearing bit 9 (the dirty flag) if accessed and dirty flags for EPT will be enabled.
- Software should use the INVEPT instruction with the "single-context" INVEPT type before a VM entry with an EPTP value X such that X[6] = 1 (accessed and dirty flags for EPT are enabled) if the logical processor had earlier been in VMX non-root operation with an EPTP value Y such that Y[6] = 0 (accessed and dirty flags for EPT are not enabled) and Y[51:12] = X[51:12].
- Software may use the INVEPT instruction after modifying a present EPT paging-structure entry (see Section 28.3.2) to change any of the privilege bits 2:0 from 0 to 1.[2] Failure to do so may cause an EPT violation that would not otherwise occur. Because an EPT violation invalidates any mappings that would be used by the access that caused the EPT violation (see Section 28.4.3.1), an EPT violation will not recur if the original access is performed again, even if the INVEPT instruction is not executed.
- Because a logical processor does not cache any information derived from EPT paging-structure entries that are not present (see Section 28.3.2) or misconfigured (see Section 28.3.3.1), it is not necessary to execute INVEPT following modification of an EPT paging-structure entry that had been not present or misconfigured.
- As detailed in Section 29.4.5, an access to the APIC-access page might not cause an APIC-access VM exit if software does not properly invalidate information that may be cached from the EPT paging structures. If EPT was in use on a logical processor at one time with EPTP X, it is recommended that software use the INVEPT instruction with the "single-context" INVEPT type and with EPTP X in the INVEPT descriptor before a VM entry on the same logical processor that enables EPT with EPTP X and either (a) the "virtualize APIC accesses" VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.
- Software can use the INVEPT instruction with the "all-context" INVEPT type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from EPT paging structures between separate uses of VMX operation.

1. If the "mode-based execute control for EPT" VM-execution control is 1, software should use the INVEPT instruction after changing privilege bit 10 from 1 to 0.
2. If the "mode-based execute control for EPT" VM-execution control is 1, software may use the INVEPT instruction after modifying a present EPT paging-structure entry to change privilege bit 10 from 0 to 1.

In a system containing more than one logical processor, software must account for the fact that information from an EPT paging-structure entry may be cached on logical processors other than the one that modifies that entry. The process of propagating the changes to a paging-structure entry is commonly referred to as "TLB shootdown." A discussion of TLB shootdown appears in Section 4.10.5, "Propagation of Paging-Structure Changes to Multiple Processors," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

24. Updates to Chapter 32, Volume 3C

Change bars and green 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: Added new Section 32.2.4, "Event Tracing". Update to Table 32-6, "IA32_RTIT_CTL MSR". Updated table name for Table 32-10, "TSX Packet Scenarios with BranchEn=1". Updates to Table 32-11, "CPUID Leaf 14H Enumeration of Intel Processor Trace Capabilities". Added new Section 32.3.9, "TNT Disable". Update to Table 32-17, "TNT Packet Definition". Update to Table 32-27, "MODE.Exec Packet Definition". Added new Section 32.4.2.29, "Control Flow Event (CFE) Packet". Added new Section 32.4.2.30, "Event Data (EVD) Packet". Updates to Section 32.5.1, "VMX-Specific Packets and VMCS Controls". Updates to Table 32-54, "Packets on VMX Transitions (System-Wide Tracing)". Update to Table 32-55, "Packets on a Failed VM Entry". Updates to Section 32.6, "Tracing and SMM Transfer Monitor (STM)", and Section 32.7, "Packet Generation Scenarios". Typo corrections throughout the chapter.

CHAPTER 32
INTEL® PROCESSOR TRACE

32.1 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.

32.1.1 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 19.5.5.2.

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 32.3. Details of the MSRs for configuring Intel PT are described in Section 32.2.8.

32.1.1.1 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 32.4):

- 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 32.4.2.2.
  - 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 *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.

32.2 INTEL® PROCESSOR TRACE OPERATIONAL MODEL

This section describes the overall Intel Processor Trace mechanism and the essential concepts relevant to how it operates.

32.2.1 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 32-1 lists branch instruction by COFI types. For detailed description of specific instructions, see *Intel® 64 and IA-32 Architectures Software Developer's Manual*.

Table 32-1. 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	INT1, INT3, INT *n*, INTO, IRET, IRETD, IRETQ, JMP (EA xx, FF /5), CALL (9A xx, FF /3), RET (CB, CA xx), SYSCALL, SYSRET, SYSENTER, SYSEXIT, VMLAUNCH, VMRESUME

32.2.1.1 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 32.2.6).

32.2.1.2 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 32.3.1.1).

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 32.4.2.2.

32.2.1.3 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 32-23 indicates exactly which IP will be included in the FUP generated by a far transfer.

32.2.2 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 *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 32-40 "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 32-6). 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.

32.2.3 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.

32.2.4 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 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 32.4.2 for packet details. Support for Event Trace is introduced in Intel® processors based on the Gracemont microarchitecture.

32.2.5 Trace Filtering

Intel Processor Trace provides filtering capabilities, by which the debug/profile tool can control what code is traced.

32.2.5.1 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 32.4.2.5).

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 32.2.6.1 for details.

32.2.5.2 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 32.3.5, "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 32.2.6.3 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 32.5.1.

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.

32.2.5.3 Filtering by IP

Trace packet generation with configurable filtering by IP is supported if CPUID.(EAX=14H, ECX=0):EBX[bit 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 32.2.8.2), 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 32.2.8.5). 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 32.3.1) 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 32.3.1), 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 32.2.6.5 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 32.3.1.1 for details. Software can query CPUID to determine filters are based on linear or effective address (Section 32.3.1).

Note that some packets, such as MTC (Section 32.3.7) 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 32.4.1.

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 32.3.8), 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 32.2.7.2.

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 32-2. IP Filtering Packet Example

Code Flow	Packets

`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`
	TIP.PGE(RangeBase)
	TNT(0)
	TIP.PGD(RangeLimit+1)

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.

32.2.6 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.

32.2.6.1 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 bookkeeping packets) may still be sent. See Section 32.2.7 for details of PacketEn and packet generation.

Note that, on processors that do not support IP filtering (i.e., CPUID.(EAX=14H, ECX=0):EBX[bit 2] = 0), FilterEn is treated as always set.

32.2.6.2 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 32.3.10).

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.

32.2.6.3 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_ENCLAVE[1]) 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 32.4.2.16 and Section 32.4.2.17), may still be generated while ContextEn is 0. For details of which packets are generated only when ContextEn is set, see Section 32.4.1.

The processor does not update ContextEn when TriggerEn = 0.

The value of ContextEn will toggle only when TriggerEn = 1.

32.2.6.4 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.

32.2.6.5 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 32.2.5.3.

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 32.4.1.

---

1. Trace packets generation is disabled in a production enclave, see Section 32.2.9.5. See *Intel® Software Guard Extensions Programming Reference* about differences between a production enclave and a debug enclave.

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.(EAX=14H, ECX=0):EBX[bit 2] = 0). FilterEn will toggle only when TraceEn=1 and ContextEn=1, and when at least one range is configured for IP filtering.

32.2.7 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 32.2.8.2):

- 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.

32.2.7.1 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 32.2.8.7) and mask value in IA32_RTIT_OUTPUT_MASK_PTRS (Section 32.2.8.8). 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 32.3.10) 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 32.2.7.4.

32.2.7.2 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 32-1 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 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 32-1 provides an illustration (not to scale) of the table and associated pointers.

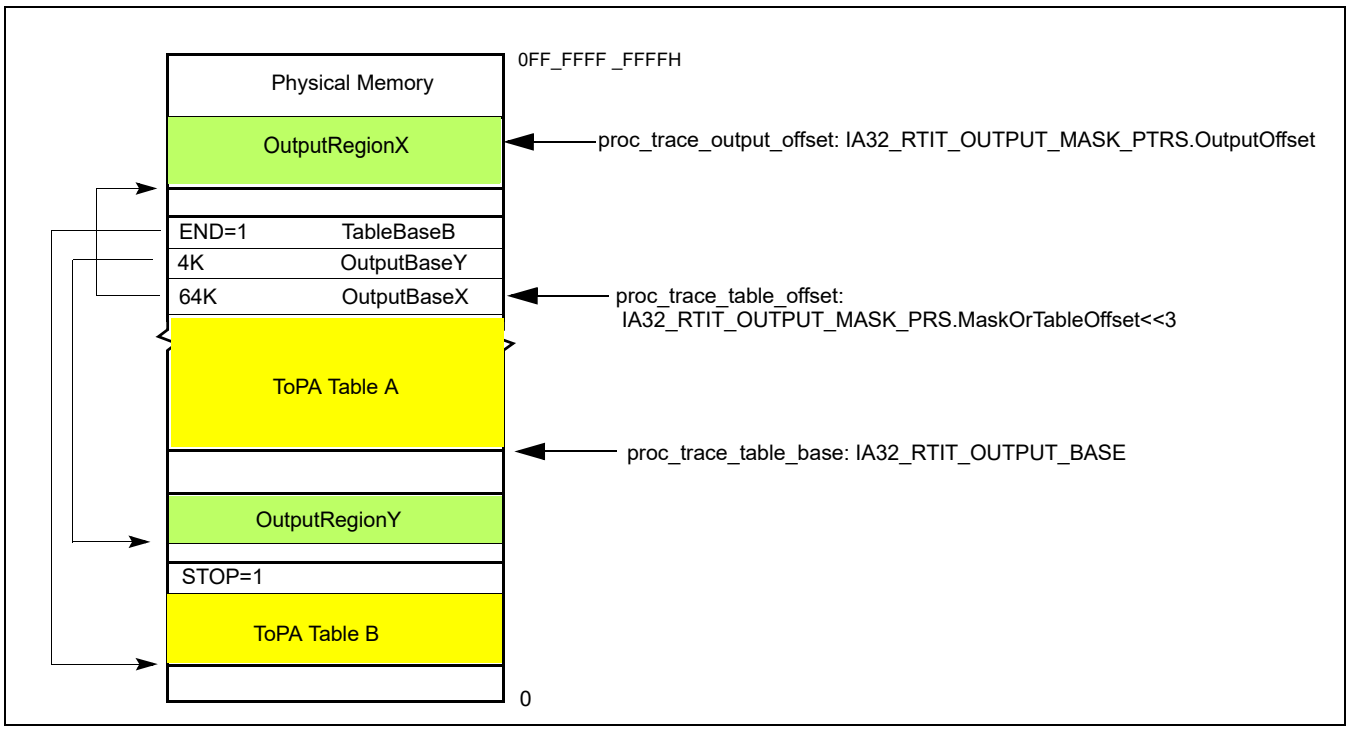


Figure 32-1. ToPA Memory Illustration

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 32-3 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.(EAX=14H,ECX=0):ECX.MENTRY[bit 1] = 0 and CPUID.(EAX=14H,ECX=0):ECX.TOPAOUT[bit 0] = 1.

If CPUID.(EAX=14H,ECX=0):ECX.MENTRY[bit 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 32-2. The size of the address field is determined by the processor's physical-address width (MAXPHYADDR) in bits, as reported in CPUID.80000008H:EAX[7:0].

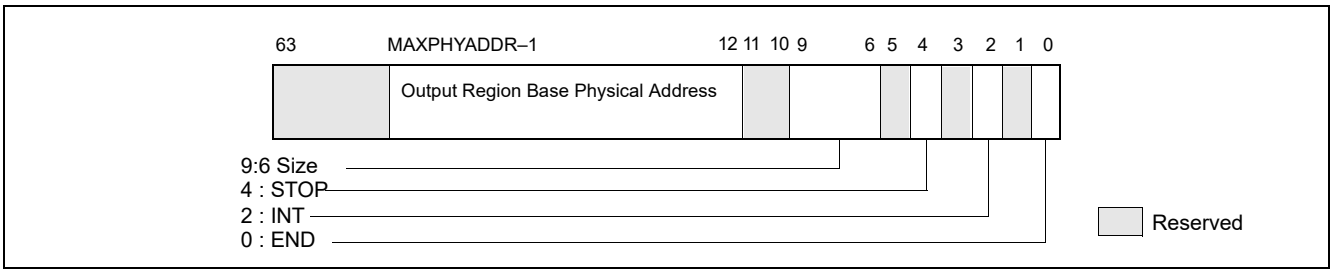


Figure 32-2. Layout of ToPA Table Entry

Table 32-3 describes the details of the ToPA table entry fields. If reserved bits are set to 1, an error is signaled.

Table 32-3. 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.

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.

Table 32-3. ToPA Table Entry Fields (Contd.)

ToPA Entry Field	Description
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 32.2.8.4 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.MaskOrTableOffset 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 *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 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.(EAX=14H, ECX=0):EBX[bit 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 32-3); 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.(EAX=14H, ECX=0):EBX[bit 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 32-4 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 32-4. Algorithm to Manage Intel PT ToPA PMI and XSAVES/XRSTORS

Pseudo Code Flow

```
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"l 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;
```

ToPA Errors

When a malformed ToPA entry is found, an **operational error** results (see Section 32.3.10). A malformed entry can be any of the following:

1. **ToPA entry reserved bit violation**.
   This describes cases where a bit marked as reserved in Section 32.2.7.2 above is set to 1.
2. **ToPA alignment violation**.
   This includes cases where illegal ToPA entry base address bits are set to 1:
   a. 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.
   b. 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.
   c. ToPA entry base address sets upper physical address bits not supported by the processor.

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 32.2.7.4 for details.

A tracing software have a range of flexibility using ToPA to manage the interaction of Intel PT with application buffers, see Section 32.4.2.26.

32.2.7.3 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(EAX=14H,ECX=0):EBX[bit 3] = 1.

32.2.7.4 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 regions. If there is any overlap with these regions, trace data collection will not function properly. Exact processor behavior is implementation-dependent; Table 32-5 summarizes several scenarios.

Table 32-5. Behavior on Restricted Memory Access

Scenario	Description
ToPA output region overlaps with SMRR	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 32.3.10) 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	The processor will signal an error (Section 32.3.10) 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 *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. No error is signaled for this case.

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.

32.2.8 Enabling and Configuration MSRs

32.2.8.1 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 32.3.1), 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.(EAX=14H, ECX=0):EBX[bit 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 32.3.5.2).

32.2.8.2 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 32-6.

Table 32-6. 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 32.2.8.3 for details of enabling and disabling packet generation. Note that the processor will clear this bit on #SMI (Section 32.2.9.3) 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 32.4.2.14). 1: Enables CYC Packet. This bit is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 1] = 0.
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 32.2.3, "Power Event Tracing").

Table 32-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
5	FUPonPTW	0	0: PTW packets are not followed by FUPs. 1: PTW packets are followed by FUPs. This bit is reserved when CPUID.(EAX=14H, ECX=0):EBX[bit 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.(EAX=14H, ECX=0):ECX[bit 3] = 0.
7	CR3Filter	0	0: Disables CR3 filtering. 1: Enables CR3 filtering. This bit is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 0] ("CR3 Filtering Support") is 0.
8	ToPA	0	0: Single-range output scheme enabled if CPUID.(EAX=14H, ECX=0):ECX.SNGLRGNOUT[bit 2] = 1 and IA32_RTIT_CTL.FabricEn=0. 1: ToPA output scheme enabled (see Section 32.2.7.2) if CPUID.(EAX=14H, ECX=0):ECX.TOPA[bit 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.(EAX=14H, ECX=0):ECX.SNGLRGNOUT[bit 2] = 0. WRMSR to IA32_RTIT_CTL that sets this bit causes #GP, if CPUID.(EAX=14H, ECX=0):ECX.TOPA[bit 0] = 0.
9	MTCEn	0	0: Disables MTC Packet (see Section 32.4.2.16). 1: Enables MTC Packet. This bit is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 3] = 0.
10	TSCEn	0	0: Disable TSC packets. 1: Enable TSC packets (see Section 32.4.2.11).
11	DisRETC	0	0: Enable RET compression. 1: Disable RET compression (see Section 32.2.1.2).
12	PTWEn	0	0: PTWRITE packet generation disabled. 1: PTWRITE packet generation enabled (see Table 32-40 "PTW Packet Definition"). This bit is reserved when CPUID.(EAX=14H, ECX=0):EBX[bit 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 32.2.6.4 for details on BranchEn.
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 32.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 3] = 0.
18	Reserved	0	Must be 0.

Table 32-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
22:19	CycThresh	0	CYC packet threshold, see Section 32.3.6 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 $2^{(CycThresh-1)}$. 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 32.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 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_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 32.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.(EAX=14H, ECX=0):EBX[bit 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.(EAX=14H, ECX=0):EBX[bit 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 32.2.5.3 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.(EAX=14H, ECX=1):EBX.RANGECNT[2:0] < 1.
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 32.2.5.3 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 32.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.(EAX=14H, ECX=1):EBX.RANGECNT[2:0] < 2.

Table 32-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
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 32.2.5.3 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 32.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.(EAX=14H, ECX=1):EBX.RANGECNT[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 32.2.5.3 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 32.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.(EAX=14H, ECX=1):EBX.RANGECNT[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.(EAX=14H, ECX=0):EBX[bit 8] ("TNT Disable Supported") is 0. SeeSection 32.3.9 for details.
56	InjectPsbPmiOnEnable	0	1: Enables use of IA32_RTIT_STATUS bits PendPSB[6] and PendTopaPMI[7], see Section 32.2.8.4, "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.(EAX=14H, ECX=0):EBX[bit 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.

32.2.8.3 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 32.4.2.17) 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 32.4.1.1).

In addition to the packets discussed above, if and when PacketEn (Section 32.2.6.1) transitions from 0 to 1 (which may happen immediately, depending on filtering settings), a TIP.PGE packet (Section 32.4.2.3) 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 config-

uration of restricted memory regions). See Section 32.7 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.

32.2.8.4 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 32-7 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 32-7. 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 32.2.6.5. Writes are ignored.
1	ContextEn	0	The processor sets this bit to indicate that tracing is allowed for the current context. See Section 32.2.6.3. Writes are ignored.
2	TriggerEn	0	The processor sets this bit to indicate that tracing is enabled. See Section 32.2.6.2. 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 32.2.7.2. 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.
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 32.2.7.2. 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.(EAX=14H, ECX=0):EBX[bit 6] = 0.

Table 32-7. IA32_RTIT_STATUS MSR

Position	Bit Name	At Reset	Bit Description
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.(EAX=14H, ECX=0):EBX[bit 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 32.4.2.17 for details. This field is reserved when CPUID.(EAX=14H,ECX=0):EBX[bit 1] ("Configurable PSB and CycleAccurate Mode Supported") is 0.
63:49	Reserved	0	Must be 0.

32.2.8.5 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 32.2.8.2). The number of these register pairs is enumerated by CPUID.(EAX=14H, ECX=1):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).

32.2.8.6 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 32.2.5.2.

This MSR is accessible if CPUID.(EAX=14H, ECX=0):EBX[bit 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.

32.2.8.7 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 the maximum physical address width (MAXPHYADDR), as reported by CPUID.80000008H:EAX[7:0].

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.(EAX=14H, ECX=0):ECX[bit 0] or CPUID.(EAX=14H, ECX=0):ECX[bit 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 32-8. 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 32.2.8.8) overlap with 1s in the base address. If the base is not aligned, an operational error will result (see Section 32.3.10). 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 32.2.7.2 as well as Section 32.3.10.
63:MAXPHYADDR	Reserved	0	Must be 0.

32.2.8.8 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 32.2.7.2 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.(EAX=14H, ECX=0):ECX[bit 0] or CPUID.(EAX=14H, ECX=0):ECX[bit 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 32-9. 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 32.3.10) 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 32.3.10) 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 32.3.10) will be signaled when TraceEn is set.

32.2.9 Interaction of Intel® Processor Trace and Other Processor Features

32.2.9.1 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 32-10 for details.

Table 32-10. 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)

Table 32-10. TSX Packet Scenarios with BranchEn=1

TSX Event	Instruction	Packets
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.

32.2.9.2 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.

32.2.9.3 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 32.2.8.2).
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 32.7 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 32.3.1.2), 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 32.6.

32.2.9.4 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 32.4.2.26.

32.2.9.5 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 32.7.

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.

32.2.9.6 SENTER/ENTERACCS and ACM

GETSEC[SENTER] and GETSEC[ENTERACCS] instructions clear TraceEn, and it is not restored when those instruction 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.

32.2.9.7 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.

32.3 CONFIGURATION AND PROGRAMMING GUIDELINE

32.3.1 Detection of Intel Processor Trace and Capability Enumeration

Processor support for Intel Processor Trace is indicated by CPUID.(EAX=07H,ECX=0H):EBX[bit 25] = 1. CPUID function 14H is dedicated to enumerate the resource and capability of processors that report CPUID.(EAX=07H,ECX=0H):EBX[bit 25] = 1. Different processor generations may have architecturally-defined variation in capabilities. Table 32-11 describes details of the enumerable capabilities that software must use across generations of processors that support Intel Processor Trace.

Table 32-11. CPUID Leaf 14H Enumeration of Intel Processor Trace Capabilities

CPUID.(EAX=14H,ECX=0)		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 32.2.8. 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 32.2.8. 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.(EAX=14H, ECX=1):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 32.2.8. 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 32.2.8. 0: An attempt to set IA32_RTIT_CTL.MTCEn or IA32_RTIT_CTL.MTCFreq to a non-zero value will generate a #GP exception.
	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.

Table 32-11. CPUID Leaf 14H Enumeration of Intel Processor Trace Capabilities (Contd.)

CPUID.(EAX=14H,ECX=0)		Name	Description Behavior
Register	Bits		
	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 32.2.7.2) IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed. 0: Unless CPUID.(EAX=14H, ECX=0):ECX.SNGLRNGOUT[bit 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 32.2.7.2. 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 32.2.7.2).
	2	Single-Range Output Supported	1: Enabling tracing (TraceEn=1) with IA32_RTIT_CTL.ToPA=0 is supported. 0: Unless CPUID.(EAX=14H, ECX=0):ECX.TOPAOUT[bit 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	
	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.(EAX=14H, ECX=0):EAX reports a non-zero value, additional capabilities of Intel Processor Trace are described in the sub-leaves of CPUID leaf 14H.

Table 32-12. CPUID Leaf 14H, sub-leaf 1H Enumeration of Intel Processor Trace Capabilities

CPUID.(EAX=14H,ECX=1)		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.(EAX=14H, ECX=0):EBX[bit 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_CTLMTCFreq 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.(EAX=14H, ECX=0):EBX[bit 1] = 1 (Cycle-Accurate Mode is Supported), otherwise the CycThresh field is reserved to 0. See Section 32.2.8. 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.
	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.(EAX=14H, ECX=0):EBX[bit 1] = 1 (Configurable PSB is supported), otherwise the PSBFreq field is reserved to 0. See Section 32.2.8. 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	

32.3.1.1 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.(EAX=14H, ECX=0):ECX.LIP[bit 31] in Table 32-11).

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.(EAX=14H, ECX=0):ECX.LIP[bit 31]. Care should be taken to check CPUID when configuring IP filters.

32.3.1.2 Model Specific Capability Restrictions

Some processor generations impose restrictions that prevent use of LBRs/BTS/BTM/LERs when software has enabled tracing with Intel Processor Trace. On these processors, when TraceEn is set, updates of LBR, BTS, BTM, LERs 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/LERs 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 processor with CPUID DisplayFamily_DisplayModel signature of 06_3DH, 06_47H, 06_4EH, 06_4FH, 06_56H and 06_5EH, the use of Intel PT and LBRs are mutually exclusive.

32.3.2 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.(EAX=07H,ECX=0H):EBX[bit 25] = 1; followed by enumerating the capabilities described in Section 32.3.1.

Based on the capability queried from Section 32.3.1, software must configure a number of model-specific registers. This section describes programming considerations related to those MSRs.

32.3.2.1 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 32.2.8.2 for details of packets generated by setting TraceEn to 1.

If setting TraceEn to 1 causes an operational error (see Section 32.3.10), 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.

32.3.2.2 Disabling Packet Generation

After disabling packet generation by clearing IA32_RTIT_CTL, it is advisable to read the IA32_RTIT_STATUS MSR (Section 32.2.8.4):

- 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 32.3.10. 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 32.2.5.3) or the ToPA Stop condition (see "ToPA STOP" in Section 32.2.7.2) before packet generation was disabled.

32.3.3 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 32.2.8.2).

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.

32.3.4 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).

32.3.5 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.

32.3.5.1 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.

32.3.5.2 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*.

32.3.6 Cycle-Accurate Mode

Intel PT can be run in a cycle-accurate mode which enables CYC packets (see Section 32.4.2.14) 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 32-1. 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

32.3.6.1 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.

32.3.6.2 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 32.8.3.2 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 32-2. 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

32.3.6.3 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 32.2.8.2).

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 32.3.1.

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 32-13 illustrates the threshold behavior.

Table 32-13. 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

32.3.7 Decoder Synchronization (PSB+)

The PSB packet (Section 32.4.2.17) 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 32.4.2.18). 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.
- 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.(EAX=14H, ECX=0):EBX[bit 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.(EAX=14H, ECX=0):EBX[bit 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.

32.3.8 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 32.4.2.16) 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.

32.3.8.1 Overflow Impact on Enables

The address comparisons to ADDRn ranges, for IP filtering and TraceStop (Section 32.2.5.3), 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.

32.3.8.2 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.

32.3.9 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].

32.3.10 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 32.2.7.2 for details on ToPA errors, and Section 32.2.7.4 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.

32.4 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.

32.4.1 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 32.4.1.1 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 32-14; see Section 32.4.1.1 for more per-packet details and Section 32.7 for more detailed packet generation examples.

Table 32-14. 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.

32.4.1.1 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 32.4.2.26) 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 32-15. Packets Forbidden Between BBP and BEP

TNT
TIP, TIP.PGE, TIP.PGD
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.

32.4.2 Packet Definitions

The following description of packet definitions are in tabular format. Figure 32-3 explains how to interpret them. Packet bits listed as "RSVD" are not guaranteed to be 0.

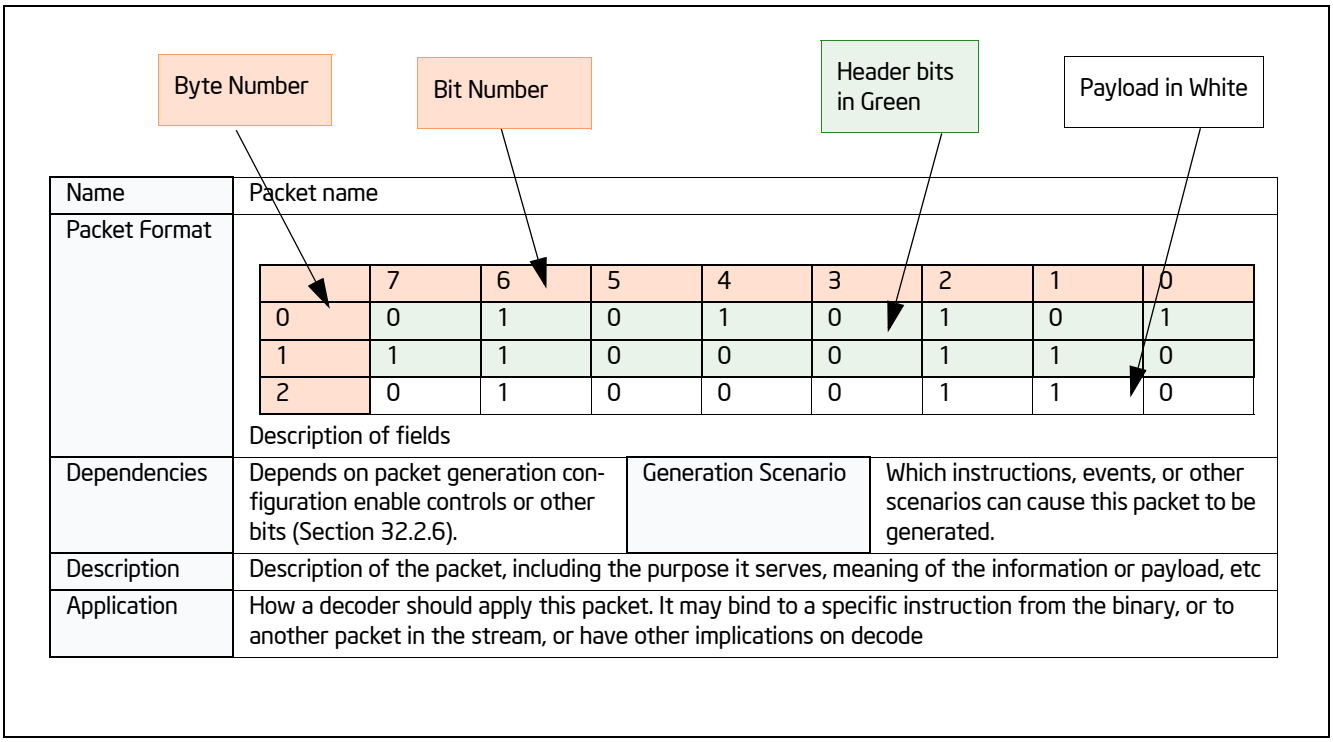


Figure 32-3. Interpreting Tabular Definition of Packet Format

32.4.2.1 Taken/Not-taken (TNT) Packet

Table 32-16. TNT Packet Definition

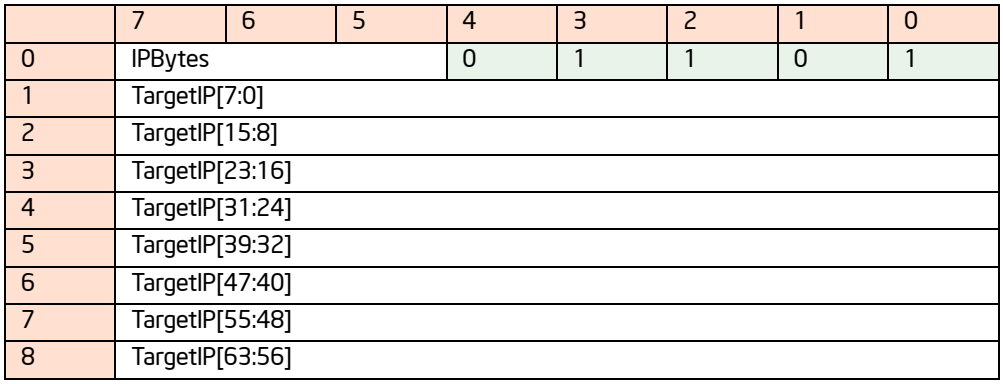
<table>
<tr><td>Name</td><td colspan="3">Taken/Not-taken (TNT) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="3">
<table>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td></td></tr>
<tr><td>0</td><td>1</td><td>B_1</td><td>B_2</td><td>B_3</td><td>B_4</td><td>B_5</td><td>B_6</td><td>0</td><td>Short TNT</td></tr>
</table>
</td></tr>
<tr><td></td><td colspan="3">B1...BN represent the last N conditional branch or compressed RET (Section 32.4.2.2) 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.</td></tr>
<tr><td></td><td colspan="3">
<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><td></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><td rowspan="8">Long TNT</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>1</td></tr>
<tr><td>2</td><td>B_{40}</td><td>B_{41}</td><td>B_{42}</td><td>B_{43}</td><td>B_{44}</td><td>B_{45}</td><td>B_{46}</td><td>B_{47}</td></tr>
<tr><td>3</td><td>B_{32}</td><td>B_{33}</td><td>B_{34}</td><td>B_{35}</td><td>B_{36}</td><td>B_{37}</td><td>B_{38}</td><td>B_{39}</td></tr>
<tr><td>4</td><td>B_{24}</td><td>B_{25}</td><td>B_{26}</td><td>B_{27}</td><td>B_{28}</td><td>B_{29}</td><td>B_{30}</td><td>B_{31}</td></tr>
<tr><td>5</td><td>B_{16}</td><td>B_{17}</td><td>B_{18}</td><td>B_{19}</td><td>B_{20}</td><td>B_{21}</td><td>B_{22}</td><td>B_{23}</td></tr>
<tr><td>6</td><td>B_8</td><td>B_9</td><td>B_{10}</td><td>B_{11}</td><td>B_{12}</td><td>B_{13}</td><td>B_{14}</td><td>B_{15}</td></tr>
<tr><td>7</td><td>1</td><td>B_1</td><td>B_2</td><td>B_3</td><td>B_4</td><td>B_5</td><td>B_6</td><td>B_7</td></tr>
</table>
</td></tr>
<tr><td></td><td colspan="3">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.
<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><td></td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>B_1</td><td>B_2</td><td>B_3</td><td>B_4</td><td>0</td><td>Short TNT</td></tr>
</table>
</td></tr>
<tr><td></td><td colspan="3">
<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><td></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><td>Long TNT</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>1</td><td></td></tr>
<tr><td>2</td><td>B_{24}</td><td>B_{25}</td><td>B_{26}</td><td>B_{27}</td><td>B_{28}</td><td>B_{29}</td><td>B_{30}</td><td>B_{31}</td><td></td></tr>
<tr><td>3</td><td>B_{16}</td><td>B_{17}</td><td>B_{18}</td><td>B_{19}</td><td>B_{20}</td><td>B_{21}</td><td>B_{22}</td><td>B_{23}</td><td></td></tr>
<tr><td>4</td><td>B_8</td><td>B_9</td><td>B_{10}</td><td>B_{11}</td><td>B_{12}</td><td>B_{13}</td><td>B_{14}</td><td>B_{15}</td><td></td></tr>
<tr><td>5</td><td>1</td><td>B_1</td><td>B_2</td><td>B_3</td><td>B_4</td><td>B_5</td><td>B_6</td><td>B_7</td><td></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>0</td><td>0</td><td></td></tr>
<tr><td>7</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>PacketEn && ~IA32_RTIT_CTL.DisTNT</td><td>Generation Scenario</td><td>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.</td></tr>
</table>

Table 32-16. 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 32.4.2.2). 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.

32.4.2.2 Target IP (TIP) Packet

Table 32-17. IP Packet Definition

Name	Target IP (TIP) Packet		
Packet Format			
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 32-18. 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 32-18), 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_RTIT_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 32.4.2.3), 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.

32.4.2.3 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 32-19. 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)

32.4.2.4 Packet Generation Enable (TIP.PGE) Packet

Table 32-20. TIP.PGE Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Target IP - Packet Generation Enable (TIP.PGE) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<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>
</td></tr>
<tr><td>Dependencies</td><td>PacketEn transitions to 1</td><td>Generation Scenario</td><td colspan="2">Any branch instruction, control flow transfer, or MOV CR3 that sets PacketEn, a WRMSR that enables packet generation and sets PacketEn</td></tr>
<tr><td>Description</td><td colspan="4">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 32.2.5.3. 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.</td></tr>
<tr><td>Application</td><td colspan="4">TIP.PGE packets bind to the instruction at the IP given in the payload.</td></tr>
</table>

32.4.2.5 Packet Generation Disable (TIP.PGD) Packet

Table 32-21. TIP.PGD Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Target IP - Packet Generation Disable (TIP.PGD) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<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>
</td></tr>
<tr><td>Dependencies</td><td>PacketEn transitions to 0</td><td>Generation Scenario</td><td colspan="2">Any branch instruction, control flow transfer, or MOV CR3 that clears PacketEn, a WRMSR that disables packet generation and clears PacketEn</td></tr>
<tr><td>Description</td><td colspan="4">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 32.2.5.3. 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 32.2.5.3 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.</td></tr>
<tr><td>Application</td><td colspan="4">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.</td></tr>
</table>

32.4.2.6 Flow Update (FUP) Packet

Table 32-22. FUP Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Flow Update (FUP) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<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>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>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn. (Typically depends on BranchEn and FilterEn as well, see Section 32.2.5, Section 32.4.2.21, and Section 32.4.2.22 for details.)</td><td>Generation Scenario</td><td colspan="2">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 n, a WRMSR that disables packet generation.</td></tr>
<tr><td>Description</td><td colspan="4">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.</td></tr>
<tr><td>Application</td><td colspan="4">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 32.4.2.8 for details.
Otherwise, FUPs are part of compound packet events (see Section 32.4.1). 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.</td></tr>
</table>

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 32-23 illustrates cases where FUPs are sent, and which IP can be expected in those cases.

Table 32-23. 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	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.

32.4.2.7 Paging Information (PIP) Packet

Table 32-24. PIP Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Paging Information (PIP) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<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>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>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td colspan="7">CR3[11:5] or 0</td><td>RSVD/NR</td></tr>
<tr><td>3</td><td colspan="8">CR3[19:12]</td></tr>
<tr><td>4</td><td colspan="8">CR3[27:20]</td></tr>
<tr><td>5</td><td colspan="8">CR3[35:28]</td></tr>
<tr><td>6</td><td colspan="8">CR3[43:36]</td></tr>
<tr><td>7</td><td colspan="8">CR3[51:44]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn && IA32_RTIT_CTL.OS</td><td>Generation Scenario</td><td colspan="2">MOV CR3, Task switch, INIT, SIPI, PSB+, VM exit, VM entry</td></tr>
<tr><td>Description</td><td colspan="4">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 paging[1]), 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 32.5.1)
▪ 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 32.2.9.3 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 32.5.1). All other PIPs clear the NR bit.</td></tr>
<tr><td>Application</td><td colspan="4">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 32.4.1). 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 32.7.</td></tr>
</table>

NOTES:

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

32.4.2.8 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 32-25. 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 32-26. MODE.Exec Packet Definition

<table>
<tr><td>Name</td><td colspan="3">MODE.Exec Packet</td></tr>
<tr><td>Packet Format</td><td colspan="3">
<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>0</td><td colspan="2">Reserved</td><td>IF</td><td>CS.D</td><td>(CS.L & LMA)</td></tr>
</table>
MODE Leaf ID is '000.
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn && FilterEn</td><td>Generation Scenario</td><td>Any operation that changes the CS.L, CS.D, or EFER.LMA, if IA32_RTIT_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.</td></tr>
<tr><td>Description</td><td colspan="3">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.0x14.0.EBX[6]=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.

<table>
<tr><td>CS.D</td><td>(CS.L & IA32_EFER.LMA)</td><td>Addressing Mode</td></tr>
<tr><td>1</td><td>1</td><td>N/A</td></tr>
<tr><td>0</td><td>1</td><td>64-bit mode</td></tr>
<tr><td>1</td><td>0</td><td>32-bit mode</td></tr>
<tr><td>0</td><td>0</td><td>16-bit mode</td></tr>
</table>
MODE.Exec packets are generated on interrupt flag (RFLAGS.IF) changes only if event tracing is enabled (EventEn=1).
</td></tr>
<tr><td>Application</td><td colspan="3">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.</td></tr>
</table>

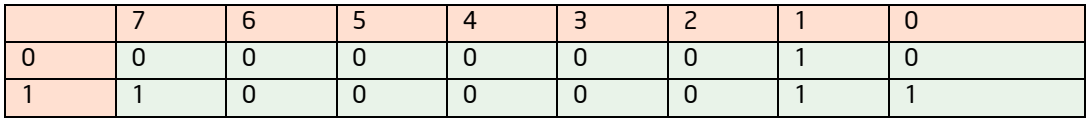
MODE.TSX Packet

Table 32-27. MODE.TSX Packet Definition

<table>
<tr><td>Name</td><td colspan="4">MODE.TSX Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<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>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn</td><td>Generation Scenario</td><td colspan="2">XBEGIN, XEND, XABORT, XACQUIRE, XRELEASE, if InTX changes, Asynchronous TSX Abort, PSB+</td></tr>
<tr><td>Description</td><td colspan="4">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.
<table>
<tr><td>TXAbort</td><td>InTX</td><td>Implication</td></tr>
<tr><td>1</td><td>1</td><td>N/A</td></tr>
<tr><td>0</td><td>1</td><td>Transaction begins, or executing transactionally</td></tr>
<tr><td>1</td><td>0</td><td>Transaction aborted</td></tr>
<tr><td>0</td><td>0</td><td>Transaction committed, or not executing transactionally</td></tr>
</table>
</td></tr>
<tr><td>Application</td><td colspan="4">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.</td></tr>
</table>

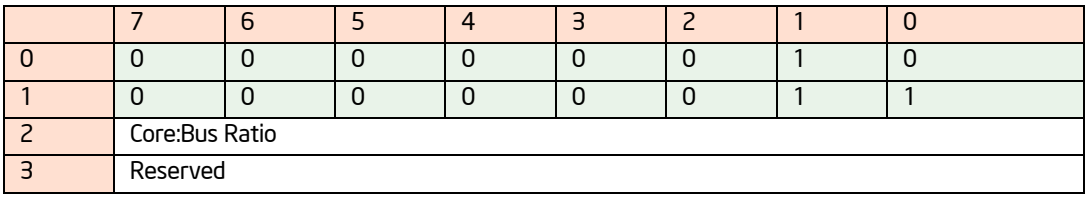
32.4.2.9 TraceStop Packet

Table 32-28. TraceStop Packet Definition

Name	TraceStop Packet		
Packet Format			
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 32.2.5.3 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.		

32.4.2.10 Core:Bus Ratio (CBR) Packet

Table 32-29. CBR Packet Definition

Name	Core:Bus Ratio (CBR) Packet		
Packet Format			
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.		

32.4.2.11 Timestamp Counter (TSC) Packet

Table 32-30. TSC Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Timestamp Counter (TSC) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<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>
</td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.TSCEn && TriggerEn</td><td>Generation Scenario</td><td colspan="2">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.</td></tr>
<tr><td>Description</td><td colspan="4">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.</td></tr>
<tr><td>Application</td><td colspan="4">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.</td></tr>
</table>

32.4.2.12 Mini Time Counter (MTC) Packet

Table 32-31. MTC Packet Definition

<table>
<tr><td>Name</td><td colspan="3">Mini time Counter (MTC) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="3">
<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>1</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">CTC[N+7:N]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.MTCEn && TriggerEn</td><td>Generation Scenario</td><td>Periodic, based on the core crystal clock, or Always Running Timer (ART).</td></tr>
<tr><td>Description</td><td colspan="3">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 leaf 15H, as described in Section 32.8.3.
Software can select the threshold N, which determines the MTC frequency by setting the IA32_RTIT_CTL.MTCFreq field (see Section 32.2.8.2) to a supported value using the lookup enumerated by CPUID (see Section 32.3.1).
See Section 32.8.3 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.</td></tr>
<tr><td>Application</td><td colspan="3">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.</td></tr>
</table>

32.4.2.13 TSC/MTC Alignment (TMA) Packet

Table 32-32. TMA Packet Definition

<table>
<tr><td>Name</td><td colspan="4">TSC/MTC Alignment (TMA) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<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>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>
</td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.MTCEn && IA32_RTIT_CTL.TSCEn && TriggerEn</td><td>Generation Scenario</td><td colspan="2">Sent with any TSC packet.</td></tr>
<tr><td>Description</td><td colspan="4">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 32.8.3.2 for details on how to make this calculation.</td></tr>
<tr><td>Application</td><td colspan="4">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.</td></tr>
</table>

32.4.2.14 Cycle Count (CYC) Packet

Table 32-33. Cycle Count Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Cycle Count (CYC) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<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="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>
</td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.CYCEn && TriggerEn</td><td>Generation Scenario</td><td colspan="2">Can be sent at any time, though a maximum of one CYC packet is sent per core clock cycle. See Section 32.3.6 for CYC-eligible packets.</td></tr>
<tr><td>Description</td><td colspan="4">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 32.2.8.2). For details on Cycle-Accurate Mode, IPC calculation, etc, see Section 32.3.6.
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.</td></tr>
<tr><td>Application</td><td colspan="4">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.</td></tr>
</table>

32.4.2.15 VMCS Packet

Table 32-34. VMCS Packet Definition

<table>
<tr><td>Name</td><td colspan="4">VMCS Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<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>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn; Also in VMX operation.</td><td>Generation Scenario</td><td colspan="2">Generated on successful VMPTRLD, and optionally on PSB+, SMM VM exits, and VM entries that return from SMM (see Section 32-53).</td></tr>
<tr><td>Description</td><td colspan="4">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 32.5.1), a VMCS packet provides this pointer. See Section 32.6 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.</td></tr>
<tr><td>Application</td><td colspan="4">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 32.4.1). 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 32.7.</td></tr>
</table>

32.4.2.16 Overflow (OVF) Packet

Table 32-35. OVF Packet Definition

Name	Overflow (OVF) Packet		
Packet Format			
Dependencies	TriggerEn	Generation Scenario	On resolution of internal buffer overflow
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 32.3.8.		
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.		

	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

32.4.2.17 Packet Stream Boundary (PSB) Packet

Table 32-36. PSB Packet Definition

Name	Packet Stream Boundary (PSB) Packet
Packet Format	

	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

Table 32-36. PSB Packet Definition (Contd.)

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 32.2.8.2). 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 32.3.7).		
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.		

32.4.2.18 PSBEND Packet

Table 32-37. 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 32.3.7).		
Application	When a PSBEND packet is seen, the decoder should cease to treat packets as "status only".		

32.4.2.19 Maintenance (MNT) Packet

Table 32-38. MNT Packet Definition

<table>
<tr><td>Name</td><td colspan="4">Maintenance (MNT) Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<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>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>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>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>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">Implementation specific.</td></tr>
<tr><td>Description</td><td colspan="4">This packet is generated by hardware, the payload meaning is model-specific.</td></tr>
<tr><td>Application</td><td colspan="4">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.</td></tr>
</table>

32.4.2.20 PAD Packet

Table 32-39. PAD Packet Definition

<table>
<tr><td>Name</td><td colspan="4">PAD Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<table>
<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>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">Implementation specific</td></tr>
<tr><td>Description</td><td colspan="4">PAD is simply a NOP packet. Processor implementations may choose to add pad packets to improve packet alignment or for implementation-specific reasons.</td></tr>
<tr><td>Application</td><td colspan="4">Ignore PAD packets.</td></tr>
</table>

32.4.2.21 PTWRITE (PTW) Packet

Table 32-40. PTW Packet Definition

<table>
<tr><td>Name</td><td colspan="4">PTW Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<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>
</td></tr>
<tr><td></td><td colspan="4">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.</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && ContextEn && FilterEn && PTWEn</td><td>Generation Scenario</td><td colspan="2">PTWRITE Instruction</td></tr>
<tr><td>Description</td><td colspan="4">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.</td></tr>
<tr><td>Application</td><td colspan="4">Binds to the associated PTWRITE instruction. The IP of the PTWRITE will be provided by a following FUP, when PTW.IP=1.</td></tr>
</table>

32.4.2.22 Execution Stop (EXSTOP) Packet

Table 32-41. EXSTOP Packet Definition

<table>
<tr><td>Name</td><td colspan="4">EXSTOP Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<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.
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && PwrEvtEn</td><td>Generation Scenario</td><td colspan="2">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</td></tr>
<tr><td>Description</td><td colspan="4">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.</td></tr>
<tr><td>Application</td><td colspan="4">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.</td></tr>
</table>

32.4.2.23 MWAIT Packet

Table 32-42. MWAIT Packet Definition

<table>
<tr><td>Name</td><td colspan="5">MWAIT Packet</td></tr>
<tr><td>Packet Format</td><td colspan="5">
<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>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td colspan="8">MWAIT Hints[7:0]</td></tr>
<tr><td>3</td><td colspan="8">Reserved</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="6">Reserved</td><td colspan="2">EXT[1:0]</td></tr>
<tr><td>7</td><td colspan="8">Reserved</td></tr>
<tr><td>8</td><td colspan="8">Reserved</td></tr>
<tr><td>9</td><td colspan="8">Reserved</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && PwrEvtEn && ContextEn</td><td>Generation Scenario</td><td colspan="3">MWAIT, UMWAIT, or TPAUSE instructions, or I/O redirection to MWAIT, that complete without fault or VMexit.</td></tr>
<tr><td>Description</td><td colspan="5">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.</td></tr>
<tr><td>Application</td><td colspan="5">The binding for the upcoming EXSTOP packet also applies to the MWAIT packet. See Section 32.4.2.22.</td></tr>
</table>

32.4.2.24 Power Entry (PWRE) Packet

Table 32-43. PWRE Packet Definition

<table>
<tr><td>Name</td><td colspan="4">PWRE Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<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>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td>HW</td><td colspan="7">Reserved</td></tr>
<tr><td>3</td><td colspan="4">Resolved Thread C-State</td><td colspan="4">Resolved Thread Sub C-State</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && PwrEvtEn</td><td>Generation Scenario</td><td colspan="2">Transition to a C-state deeper than C0.0.</td></tr>
<tr><td>Description</td><td colspan="4">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 14.5, "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.</td></tr>
<tr><td>Application</td><td colspan="4">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.</td></tr>
</table>

32.4.2.25 Power Exit (PWRX) Packet

Table 32-44. PWRX Packet Definition

<table>
<tr><td>Name</td><td colspan="4">PWRX Packet</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<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>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn && PwrEvtEn</td><td>Generation Scenario</td><td colspan="2">Transition from a C-state deeper than C0.0 to C0.</td></tr>
<tr><td>Description</td><td colspan="4">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><th>Bit</th><th>Field</th><th>Meaning</th></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>
</td></tr>
<tr><td>Application</td><td colspan="4">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.</td></tr>
</table>

32.4.2.26 Block Begin Packet (BBP)

Table 32-45. Block Begin Packet Definition

<table>
<tr><td>Name</td><td colspan="4">BBP</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<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="3">Reserved</td><td colspan="4">Type[4:0]</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">PEBS event, if IA32_PEBS_ENABLE.OUTPUT=1.</td></tr>
<tr><td>Description</td><td colspan="4">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>
</td></tr>
<tr><td>Application</td><td colspan="4">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.</td></tr>
</table>

32.4.2.27 Block Item Packet (BIP)

Table 32-46. Block Item Packet Definition

<table>
<tr><td>Name</td><td colspan="4">BIP</td></tr>
<tr><td>Packet Format</td><td colspan="4">If the preceding BBP.SZ=0:
<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">ID[5:0]</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td colspan="8">Payload[7:0]</td></tr>
<tr><td>2</td><td colspan="8">Payload[15:8]</td></tr>
<tr><td>3</td><td colspan="8">Payload[23:16]</td></tr>
<tr><td>4</td><td colspan="8">Payload[31:24]</td></tr>
<tr><td>5</td><td colspan="8">Payload[39:32]</td></tr>
<tr><td>6</td><td colspan="8">Payload[47:40]</td></tr>
<tr><td>7</td><td colspan="8">Payload[55:48]</td></tr>
<tr><td>8</td><td colspan="8">Payload[63:56]</td></tr>
</table>
If the preceding BBP.SZ=1:
<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">ID[5:0]</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td colspan="8">Payload[7:0]</td></tr>
<tr><td>2</td><td colspan="8">Payload[15:8]</td></tr>
<tr><td>3</td><td colspan="8">Payload[23:16]</td></tr>
<tr><td>4</td><td colspan="8">Payload[31:24]</td></tr>
</table></td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">See BBP.</td></tr>
<tr><td>Description</td><td colspan="4">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 32-47 below for the complete list.</td></tr>
<tr><td>Application</td><td colspan="4">See BBP.</td></tr>
</table>

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 32-47. 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

Table 32-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x01	0x04	R/EDX
0x01	0x05	R/EBX
0x01	0x06	R/ESP
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 19.9.2.2.1)		
0x04	0x00	Instruction Pointer
0x04	0x01	Applicable Counters
0x04	0x02	Timestamp
PEBS Memory Info (Section 19.9.2.2.2)		
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

Table 32-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x08	0x0E	LBR[TOS-4]_INFO
0x08	0x0F	LBR[TOS-5]_FROM_IP
0x08	0x10	LBR[TOS-5]_TO_IP
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

Table 32-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x09	0x13	LBR[TOS-17]_FROM_IP
0x09	0x14	LBR[TOS-17]_TO_IP
0x09	0x15	LBR[TOS-17]_INFO
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

Table 32-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x0A	0x18	LBR[TOS-29]_TO_IP
0x0A	0x19	LBR[TOS-29]_INFO
0x0A	0x1A	LBR[TOS-30]_FROM_IP
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

Table 32-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x10	0x1D	XMM14_Q1
0x10	0x1E	XMM15_Q0
0x10	0x1F	XMM15_Q1

32.4.2.28 Block End Packet (BEP)

Table 32-48. Block End Packet Definition

<table>
<tr><td>Name</td><td colspan="4">BEP</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<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>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>TriggerEn</td><td>Generation Scenario</td><td colspan="2">See BBP.</td></tr>
<tr><td>Description</td><td colspan="4">Indicates the end of a packet block. The IP bit indicates if a FUP will follow, and will be set if ContextEn=1.</td></tr>
<tr><td>Application</td><td colspan="4">The block, from initial BBP to the BEP, binds to the FUP IP, if IP=1, and consumes the FUP.</td></tr>
</table>

32.4.2.29 Control Flow Event (CFE) Packet

Table 32-49. Control Flow Event Packet Definition

<table>
<tr><td>Name</td><td colspan="3">CFE</td></tr>
<tr><td>Packet Format</td><td colspan="3">
<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="4">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.
</td></tr>
<tr><td>Dependencies</td><td>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.</td><td>Generation Scenario</td><td>Software interrupt, external interrupt, or exception, including those injected on VM entry.
INIT, SIPI, SMI, RSM, 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.</td></tr>
<tr><td>Description</td><td colspan="3">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 32-50 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.</td></tr>
<tr><td>Application</td><td colspan="3">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.</td></tr>
</table>

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 32-50. 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 that address of the instruction that has not completed (NLIP for trap events, CLIP for fault events).
IRET	0x2	Invalid	CLIP	
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.

Table 32-50. CFE Packet Type and Vector Fields Details (Contd.)

CFE Subtype	Type	Vector	FUP IP	Details
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	
Reserved	0xc...0x1f	N/A	N/A	Reserved

32.4.2.30 Event Data (EVD) Packet

Table 32-51. Event Data Packet Definition

<table>
<tr><td>Name</td><td colspan="4">EVD</td></tr>
<tr><td>Packet Format</td><td colspan="4">
<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>
</td></tr>
<tr><td>Dependencies</td><td>IA32_RTIT_CTL.EventEn && TriggerEn && ContextEn</td><td>Generation Scenario</td><td colspan="2">Page fault, including those injected on VM entry.
VM exit, if “Suppress VMX packets on exit” VMCS exit control is 0.</td></tr>
<tr><td>Description</td><td colspan="4">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.</td></tr>
<tr><td>Application</td><td colspan="4">EVD packets bind to the same IP (if any) as the subsequent CFE packet.</td></tr>
</table>

32.5 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 32.5.1 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 24.7.1), “Load IA32_RTIT_CTL” entry control (See Section 24.8.1), and “Intel PT uses guest physical addresses” execution control (See Section 25.5.3).

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 32.5.2 for details. Tracing with SMM Transfer Monitor is described in Section 32.6.

32.5.1 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 32.4.2.15 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 32-52. 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 32-52 are not supported, and VM entry will fail on any attempt to set them.

32.5.2 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.

32.5.2.1 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 32-53.

Table 32-53. 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 32.4.2.8). 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 32.4.2.8). 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.(EAX=07H, ECX=0):EBX[bit 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 25, "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.

32.5.2.2 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 24.7.2, "VM-Exit Controls for MSRs") and VM entries (see Section 24.8.2, "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 32-52.

32.5.2.3 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.

32.5.2.4 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 leaf 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 leaf 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 32.8.3 for details on tracking time within an Intel PT trace.

32.5.2.5 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 32-54. 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)	*CFE.VMENTRY, FUP(CLIP) if EventEn=1* *PIP(Guest CR3, NR=1), TraceEn 0⟶1 Packets (See Section 32.2.8.3),* PIP(HostCR3, NR=0), TIP(HostIP)
VMM Only	"Load IA32_RTIT_CTL" entry control or VM-entry MSR-load area used to clear TraceEn	TIP (NextIP)	*TraceEn 0⟶1 Packets (See Section 32.2.8.3),* TIP(HostIP)
VM Only	"Load IA32_RTIT_CTL" entry control or VM-entry MSR-load area used to set TraceEn	None	None

32.5.2.6 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.

32.6 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 . 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.

32.7 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 32-55 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 32-55. 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	1st..5th 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)

Table 32-55. Packet Generation under Different Example Operations

Case	Operation	Details	Packets
15	VM exit	Assumes system-wide tracing, see Section 32.5.2.1	See Table 32-53
16	VM entry	Assumes system-wide tracing, see Section 32.5.2.1	See Table 32-53
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

Table 32-56 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 32.3.7 for details.

Table 32-56. 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()

Table 32-56. Packet Generation with Operations That Alter the Value of PacketEn (Contd.)

Case	Operation	PktEn Before	PktEn After	CntxEn After	Details	Packets
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()
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 32.5.2.2. TraceEn is cleared.	FUP(VMCSg.RIP), TIP.PGD()
12	VM entry	0	1	1	Assumes guest-only tracing, see Section 32.5.2.2. TraceEn is set.	TIP.PGE(VMCSg.RIP)

Table 32-57 illustrates examples of PTWRITE, assuming TriggerEn && PTWEn is true.

Table 32-57. 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 32-58 illustrates examples of Power Event Trace, assuming TriggerEn && PwrEvtEn is true.

Table 32-58. 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)?

Table 32-58. Examples of Power Event Trace when (Contd.)TriggerEn && PwrEvtEn is True

Case	Operation	ContextEn && FilterEn	Details	Packets
3	PTWRITE rm64	1	FUP, PTW.IP=1 if FUPonPTW=1	PTW(IP=1?, 8B, rm64_value), FUP(CLIP)?

Table 32-59 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 32-59. Event Trace Examples when TriggerEn && ContextEn && EventEn is True

Case	Operation	ContextEn Before	ContextEn After	Details	Packets
1	IRET	1	D.C.		CFE.IRET(IP=1), FUP(CLIP)
2	IRET	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, 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_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(VMCSh.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(CLIP)

32.8 SOFTWARE CONSIDERATIONS

32.8.1 Tracing SMM Code

Nothing prevents an SMM handler from configuring and enabling packet generation for its own use. As described in Section Section 32.2.9.3, 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 32.3.1.2), any RSM during which TraceEn is restored to 1 will suspend any LBR or BTS logging.

32.8.2 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.

32.8.3 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.InvariantTSC[bit 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.

32.8.3.1 Time Domain Relationships

The three domains are related by the following formula:

```
TimeStampValue = (CoreCrystalClockValue * P) + AdjustedProcessorCycles + 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 leaf 15H, as described in Section 32.8.3.

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.

32.8.3.2 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 32.8.3.1 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 leaf 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:

$(CTC_B - CTC_A)$, where $CTC_i = MTC_i[15:8]$ << IA32_RTIT_CTL.MTCFreq 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

$CTC_{Delta}[15:0] = (CTC_{Next}[15:0] - TMA.CTC[15:0])$, where $CTC_{Next} = MTC_{Payload}$ << 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 $P1/CBR_{payload}$.

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 32.8.3.4 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.

32.8.3.3 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 25, "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 32.5.2.4 for details.

32.8.3.4 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 32.8.3.1 above for details on the relationship between TSC, MTC, and CYC.

25. Updates to Chapter 37, Volume 3D

Change bars 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: Updated pseudocode in EAUG, EEXTEND, EACCEPT, and EGETKEY. Updated pseudocode in EENTER to move a group of FS/GS checks from one location to another later in the flow.

CHAPTER 37
INTEL® SGX INSTRUCTION REFERENCES

This chapter describes the supervisor and user level instructions provided by Intel® Software Guard Extensions (Intel® SGX). In general, various functionality is encoded as leaf functions within the ENCLS (supervisor), ENCLU (user), and the ENCLV (virtualization operation) instruction mnemonics. Different leaf functions are encoded by specifying an input value in the EAX register of the respective instruction mnemonic.

37.1 INTEL® SGX INSTRUCTION SYNTAX AND OPERATION

ENCLS, ENCLU and ENCLV instruction mnemonics for all leaf functions are covered in this section.

For all instructions, the value of CS.D is ignored; addresses and operands are 64 bits in 64-bit mode and are otherwise 32 bits. Aside from EAX specifying the leaf number as input, each instruction leaf may require all or some subset of the RBX/RCX/RDX as input parameters. Some leaf functions may return data or status information in one or more of the general purpose registers.

37.1.1 ENCLS Register Usage Summary

Table 37-1 summarizes the implicit register usage of supervisor mode enclave instructions.

Table 37-1. Register Usage of Privileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
ECREATE	00H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EADD	01H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EINIT	02H (In)	SIGSTRUCT (In, EA)	SECS (In, EA)	EINITTOKEN (In, EA)
EREMOVE	03H (In)		EPCPAGE (In, EA)	
EDBGRD	04H (In)	Result Data (Out)	EPCPAGE (In, EA)	
EDBGWR	05H (In)	Source Data (In)	EPCPAGE (In, EA)	
EEXTEND	06H (In)	SECS (In, EA)	EPCPAGE (In, EA)	
ELDB	07H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
ELDU	08H (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
EBLOCK	09H (In)		EPCPAGE (In, EA)	
EPA	0AH (In)	PT_VA (In)	EPCPAGE (In, EA)	
EWB	0BH (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	VERSION (In, EA)
ETRACK	0CH (In)		EPCPAGE (In, EA)	
EAUG	0DH (In)	PAGEINFO (In, EA)	EPCPAGE (In, EA)	
EMODPR	0EH (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EMODT	0FH (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
ERDINFO	010H (In)	RDINFO (In, EA*)	EPCPAGE (In, EA)	
ETRACKC	011H (In)		EPCPAGE (In, EA)	
ELDBC	012H (In)	PAGEINFO (In, EA*)	EPCPAGE (In, EA)	VERSION (In, EA)
ELDUC	013H (In)	PAGEINFO (In, EA*)	EPCPAGE (In, EA)	VERSION (In, EA)
EA: Effective Address				

37.1.2 ENCLU Register Usage Summary

Table 37-2 summarizes the implicit register usage of user mode enclave instructions.

Table 37-2. Register Usage of Unprivileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
EREPORT	00H (In)	TARGETINFO (In, EA)	REPORTDATA (In, EA)	OUTPUTDATA (In, EA)
EGETKEY	01H (In)	KEYREQUEST (In, EA)	KEY (In, EA)	
EENTER	02H (In)	TCS (In, EA)	AEP (In, EA)	
	RBX.CSSA (Out)		Return (Out, EA)	
ERESUME	03H (In)	TCS (In, EA)	AEP (In, EA)	
EEXIT	04H (In)	Target (In, EA)	Current AEP (Out)	
EACCEPT	05H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EMODPE	06H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EACCEPTCOPY	07H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	EPCPAGE (In, EA)
EA: Effective Address				

37.1.3 ENCLV Register Usage Summary

Table 37-3 summarizes the implicit register usage of virtualization operation enclave instructions.

Table 37-3. Register Usage of Virtualization Operation Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
EDECVIRTCHILD	00H (In)	EPCPAGE (In, EA)	SECS (In, EA)	
EINCVIRTCHILD	01H (In)	EPCPAGE (In, EA)	SECS (In, EA)	
ESETCONTEXT	02H (In)		EPCPAGE (In, EA)	Context Value (In, EA)
EA: Effective Address				

37.1.4 Information and Error Codes

Information and error codes are reported by various instruction leaf functions to show an abnormal termination of the instruction or provide information which may be useful to the developer. Table 37-4 shows the various codes and the instruction which generated the code. Details of the meaning of the code is provided in the individual instruction.

Table 37-4. Error or Information Codes for Intel® SGX Instructions

Name	Value	Returned By
No Error	0	
SGX_INVALID_SIG_STRUCT	1	EINIT
SGX_INVALID_ATTRIBUTE	2	EINIT, EGETKEY
SGX_BLKSTATE	3	EBLOCK
SGX_INVALID_MEASUREMENT	4	EINIT
SGX_NOTBLOCKABLE	5	EBLOCK
SGX_PG_INVLD	6	EBLOCK, ERDINFO, ETRACKC
SGX_EPC_PAGE_CONFLICT	7	EBLOCK, EMODPR, EMODT, ERDINFO , EDECVIRTCHILD, EINCVIRTCHILD, ELDBC, ELDUC, ESETCONTEXT, ETRACKC

Table 37-4. Error or Information Codes for Intel® SGX Instructions

Name	Value	Returned By
SGX_INVALID_SIGNATURE	8	EINIT
SGX_MAC_COMPARE_FAIL	9	ELDB, ELDU, ELDBC, ELDUC
SGX_PAGE_NOT_BLOCKED	10	EWB
SGX_NOT_TRACKED	11	EWB, EACCEPT
SGX_VA_SLOT_OCCUPIED	12	EWB
SGX_CHILD_PRESENT	13	EWB, EREMOVE
SGX_ENCLAVE_ACT	14	EREMOVE
SGX_ENTRYEPOCH_LOCKED	15	EBLOCK
SGX_INVALID_EINITTOKEN	16	EINIT
SGX_PREV_TRK_INCMPL	17	ETRACK, ETRACKC
SGX_PG_IS_SECS	18	EBLOCK
SGX_PAGE_ATTRIBUTES_MISMATCH	19	EACCEPT, EACCEPTCOPY
SGX_PAGE_NOT_MODIFIABLE	20	EMODPR, EMODT
SGX_PAGE_NOT_DEBUGGABLE	21	EDBGRD, EDBGWR
SGX_INVALID_COUNTER	25	EDECVIRTCHILD
SGX_PG_NONEPC	26	ERDINFO
SGX_TRACK_NOT_REQUIRED	27	ETRACKC
SGX_INVALID_CPUSVN	32	EINIT, EGETKEY
SGX_INVALID_ISVSVN	64	EGETKEY
SGX_UNMASKED_EVENT	128	EINIT
SGX_INVALID_KEYNAME	256	EGETKEY

37.1.5 Internal CREGs

The CREGs as shown in Table 5-4 are hardware specific registers used in this document to indicate values kept by the processor. These values are used while executing in enclave mode or while executing an Intel SGX instruction. These registers are not software visible and are implementation specific. The values in Table 37-5 appear at various places in the pseudo-code of this document. They are used to enhance understanding of the operations.

Table 37-5. List of Internal CREG

Name	Size (Bits)	Scope
CR_ENCLAVE_MODE	1	LP
CR_DBGOPTIN	1	LP
CR_TCS_LA	64	LP
CR_TCS_PA	64	LP
CR_ACTIVE_SECS	64	LP
CR_ELRANGE	128	LP
CR_SAVE_TF	1	LP
CR_SAVE_FS	64	LP
CR_GPR_PA	64	LP
CR_XSAVE_PAGE_n	64	LP
CR_SAVE_DR7	64	LP
CR_SAVE_PERF_GLOBAL_CTRL	64	LP

Table 37-5. List of Internal CREG

Name	Size (Bits)	Scope
CR_SAVE_DEBUGCTL	64	LP
CR_SAVE_PEBS_ENABLE	64	LP
CR_CPUSVN	128	PACKAGE
CR_SGXOWNEREPOCH	128	PACKAGE
CR_SAVE_XCR0	64	LP
CR_SGX_ATTRIBUTES_MASK	128	LP
CR_PAGING_VERSION	64	PACKAGE
CR_VERSION_THRESHOLD	64	PACKAGE
CR_NEXT_EID	64	PACKAGE
CR_BASE_PK	128	PACKAGE
CR_SEAL_FUSES	128	PACKAGE
CR_CET_SAVE_AREA_PA	64	LP
CR_ENCLAVE_SS_TOKEN_PA	64	LP
CR_SAVE_IA32_U_CET	64	LP
CR_SAVE_SSP	64	LP

37.1.6 Concurrent Operation Restrictions

Under certain conditions, Intel SGX disallows certain leaf functions from operating concurrently. Listed below are some examples of concurrency that are not allowed.

- For example, Intel SGX disallows the following leafs to concurrently operate on the same EPC page.
  - ECREATE, EADD, and EREMOVE are not allowed to operate on the same EPC page concurrently with themselves.
  - EADD, EEXTEND, and EINIT leaves are not allowed to operate on the same SECS concurrently.
- Intel SGX disallows the EREMOVE leaf from removing pages from an enclave that is in use.
- Intel SGX disallows entry (EENTER and ERESUME) to an enclave while a page from that enclave is being removed.

When disallowed operation is detected, a leaf function may do one of the following:

- Return an SGX_EPC_PAGE_CONFLICT error code in RAX.
- Cause a #GP(0) exception.

To prevent such exceptions, software must serialize leaf functions or prevent these leaf functions from accessing the same EPC page.

37.1.6.1 Concurrency Tables of Intel® SGX Instructions

The tables below detail the concurrent operation restrictions of all SGX leaf functions. For each leaf function, the table has a separate line for each of the EPC pages the leaf function accesses.

For each such EPC page, the base concurrency requirements are detailed as follows:

- **Exclusive Access** means that no other leaf function that requires either shared or exclusive access to the same EPC page may be executed concurrently. For example, EADD requires an exclusive access to the target page it accesses.
- **Shared Access** means that no other leaf function that requires an exclusive access to the same EPC page may be executed concurrently. Other leaf functions that require shared access may run concurrently. For example, EADD requires a shared access to the SECS page it accesses.

- **Concurrent Access** means that any other leaf function that requires any access to the same EPC page may be executed concurrently. For example, EGETKEY has no concurrency requirements for the KEYREQUEST page.

In addition to the base concurrency requirements, additional concurrency requirements are listed, which apply only to specific sets of leaf functions. For example, there are additional requirements that apply for EADD, EXTEND and EINIT. EADD and EEXTEND can't execute concurrently on the same SECS page.

The tables also detail the leaf function's behavior when a conflict happens, i.e., a concurrency requirement is not met. In this case, the leaf function may return an SGX_EPC_PAGE_CONFLICT error code in RAX, or it may cause an exception. In addition, the tables detail those conflicts where a VM Exit may be triggered, and list the Exit Qualification code that is provided in such cases.

Table 37-6. Base Concurrency Restrictions

Leaf	Parameter		Base Concurrency Restrictions		
			Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EACCEPT	Target	[DS:RCX]	Shared	#GP	
	SECINFO	[DS:RBX]	Concurrent		
EACCEPTCOPY	Target	[DS:RCX]	Concurrent		
	Source	[DS:RDX]	Concurrent		
	SECINFO	[DS:RBX]	Concurrent		
EADD	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS	[DS:RBX]PAGEINFO.SECS	Shared	#GP	
EAUG	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS	[DS:RBX]PAGEINFO.SECS	Shared	#GP	
EBLOCK	Target	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
ECREATE	SECS	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
EDBGRD	Target	[DS:RCX]	Shared	#GP	
EDBGWR	Target	[DS:RCX]	Shared	#GP	
EDECVIRTCHILD	Target	[DS:RBX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS	[DS:RCX]	Concurrent		
EENTERTCS	SECS	[DS:RBX]	Shared	#GP	
EEXIT			Concurrent		
EEXTEND	Target	[DS:RCX]	Shared	#GP	
	SECS	[DS:RBX]	Concurrent		
EGETKEY	KEYREQUEST	[DS:RBX]	Concurrent		
	OUTPUTDATA	[DS:RCX]	Concurrent		
EINCVIRTCHILD	Target	[DS:RBX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS	[DS:RCX]	Concurrent		
EINIT	SECS	[DS:RCX]	Shared	#GP	
ELDB/ELDU	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA	[DS:RDX]	Shared	#GP	
	SECS	[DS:RBX]PAGEINFO.SECS	Shared	#GP	

Table 37-6. Base Concurrency Restrictions

Leaf	Parameter		Base Concurrency Restrictions		
			Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDLBC/ELDUC	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	EPC_PAGE_CONFLICT_ERROR
	VA	[DS:RDX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS	[DS:RBX]PAGEINFO.SECS	Shared	SGX_EPC_PAGE_CONFLICT	
EMODPE	Target	[DS:RCX]	Concurrent		
	SECINFO	[DS:RBX]	Concurrent		
EMODPR	Target	[DS:RCX]	Shared	#GP	
EMODT	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	EPC_PAGE_CONFLICT_ERROR
EPA	VA	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
ERDINFO	Target	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
EREMOVE	Target	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
EREPORT	TARGETINFO	[DS:RBX]	Concurrent		
	REPORTDATA	[DS:RCX]	Concurrent		
	OUTPUTDATA	[DS:RDX]	Concurrent		
ERESUME	TCS	[DS:RBX]	Shared	#GP	
ESETCONTEXT	SECS	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
ETRACK	SECS	[DS:RCX]	Shared	#GP	
ETRACKC	Target	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS	Implicit	Concurrent		
EWB	Source	[DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA	[DS:RDX]	Shared	#GP	

Table 37-7. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
			Access	On Conflict	Access	On Conflict	Access	On Conflict
EACCEPT	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EACCEPTCOPY	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	Source	[DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	

Table 37-7. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
			Access	On Conflict	Access	On Conflict	Access	On Conflict
EADD	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Exclusive	#GP	Concurrent	
EAUG	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
EBLOCK	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
ECREATE	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EDBGRD	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EDBGWR	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EDECVIRTCHILD	Target	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EENTERTCS	SECS	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EEXIT			Concurrent		Concurrent		Concurrent	
EEXTEND	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]	Concurrent		Exclusive	#GP	Concurrent	
EGETKEY	KEYREQUEST	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EINCVIRTCHILD	Target	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EINIT	SECS	[DS:RCX]	Concurrent		Exclusive	#GP	Concurrent	
ELDB/ELDU	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
EDLBC/ELDUC	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
EMODPE	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EMODPR	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	
EMODT	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	
EPA	VA	[DS:RCX]	Concurrent		Concurrent		Concurrent	

Table 37-7. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
			Access	On Conflict	Access	On Conflict	Access	On Conflict
ERDINFO	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EREMOVE	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EREPORT	TARGETINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	REPORTDATA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA	[DS:RDX]	Concurrent		Concurrent		Concurrent	
ERESUME	TCS	[DS:RBX]	Concurrent		Concurrent		Concurrent	
ESETCONTEXT	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
ETRACK	SECS	[DS:RCX]	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT[1]
ETRACKC	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	Implicit	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT[1]
EWB	Source	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	

NOTES:

1. SGX_CONFLICT VM Exit Qualification =TRACKING_RESOURCE_CONFLICT.

37.2 INTEL® SGX INSTRUCTION REFERENCE

ENCLS—Execute an Enclave System Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 CF ENCLS	ZO	V/V	NA	This instruction is used to execute privileged Intel SGX leaf functions that are used for managing and debugging the enclaves.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
ZO	NA	NA	NA	See Section 37.3

Description

The ENCLS instruction invokes the specified privileged Intel SGX leaf function for managing and debugging enclaves. Software specifies the leaf function by setting the appropriate value in the register EAX as input. The registers RBX, RCX, and RDX have leaf-specific purpose, and may act as input, as output, or may be unused. In 64-bit mode, the instruction ignores upper 32 bits of the RAX register.

The ENCLS instruction produces an invalid-opcode exception (#UD) if CR0.PE = 0 or RFLAGS.VM = 1, or if it is executed in system-management mode (SMM). Additionally, any attempt to execute the instruction when CPL > 0 results in #UD. The instruction produces a general-protection exception (#GP) if CR0.PG = 0 or if an attempt is made to invoke an undefined leaf function.

In VMX non-root operation, execution of ENCLS may cause a VM exit if the "enable ENCLS exiting" VM-execution control is 1. In this case, execution of individual leaf functions of ENCLS is governed by the ENCLS-exiting bitmap field in the VMCS. Each bit in that field corresponds to the index of an ENCLS leaf function (as provided in EAX).

Software in VMX root operation can thus intercept the invocation of various ENCLS leaf functions in VMX non-root operation by setting the "enable ENCLS exiting" VM-execution control and setting the corresponding bits in the ENCLS-exiting bitmap.

Addresses and operands are 32 bits outside 64-bit mode (IA32_EFER.LMA = 0 || CS.L = 0) and are 64 bits in 64-bit mode (IA32_EFER.LMA = 1 || CS.L = 1). CS.D value has no impact on address calculation. The DS segment is used to create linear addresses.

Segment override prefixes and address-size override prefixes are ignored, and is the REX prefix in 64-bit mode.

Operation

```
IF TSX_ACTIVE
    THEN GOTO TSX_ABORT_PROCESSING; FI;

IF CR0.PE = 0 or RFLAGS.VM = 1 or in SMM or CPUID.SGX_LEAF.0:EAX.SE1 = 0
    THEN #UD; FI;

IF (CPL > 0)
    THEN #UD; FI;

IF in VMX non-root operation and the "enable ENCLS exiting" VM-execution control is 1
    THEN
        IF EAX < 63 and ENCLS_exiting_bitmap[EAX] = 1 or EAX> 62 and ENCLS_exiting_bitmap[63] = 1
            THEN VM exit;
        FI;
FI;
IF IA32_FEATURE_CONTROL.LOCK = 0 or IA32_FEATURE_CONTROL.SGX_ENABLE = 0
    THEN #GP(0); FI;

IF (EAX is an invalid leaf number)
    THEN #GP(0); FI;
```

```
IF CR0.PG = 0
    THEN #GP(0); FI;

(* DS must not be an expanded down segment *)
IF not in 64-bit mode and DS.Type is expand-down data
    THEN #GP(0); FI;

Jump to leaf specific flow
```

Flags Affected

See individual leaf functions

Protected Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 0.
	If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.
	If data segment expand down.
	If CR0.PG=0.

Real-Address Mode Exceptions

#UD	ENCLS is not recognized in real mode.

Virtual-8086 Mode Exceptions

#UD	ENCLS is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 0.
	If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.

ENCLU—Execute an Enclave User Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 D7 ENCLU	ZO	V/V	NA	This instruction is used to execute non-privileged Intel SGX leaf functions.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
ZO	NA	NA	NA	See Section 37.4

Description

The ENCLU instruction invokes the specified non-privileged Intel SGX leaf functions. Software specifies the leaf function by setting the appropriate value in the register EAX as input. The registers RBX, RCX, and RDX have leaf-specific purpose, and may act as input, as output, or may be unused. In 64-bit mode, the instruction ignores upper 32 bits of the RAX register.

The ENCLU instruction produces an invalid-opcode exception (#UD) if CR0.PE = 0 or RFLAGS.VM = 1, or if it is executed in system-management mode (SMM). Additionally, any attempt to execute this instruction when CPL < 3 results in #UD. The instruction produces a general-protection exception (#GP) if either CR0.PG or CR0.NE is 0, or if an attempt is made to invoke an undefined leaf function. The ENCLU instruction produces a device not available exception (#NM) if CR0.TS = 1.

Addresses and operands are 32 bits outside 64-bit mode (IA32_EFER.LMA = 0 or CS.L = 0) and are 64 bits in 64-bit mode (IA32_EFER.LMA = 1 and CS.L = 1). CS.D value has no impact on address calculation. The DS segment is used to create linear addresses.

Segment override prefixes and address-size override prefixes are ignored, as is the REX prefix in 64-bit mode.

Operation

```
IN_64BIT_MODE := 0;
IF TSX_ACTIVE
    THEN GOTO TSX_ABORT_PROCESSING; FI;

(* If enclosing app has CET indirect branch tracking enabled then if it is not ERESUME leaf cause a #CP fault *)
(* If the ERESUME is not successful it will leave tracker in WAIT_FOR_ENDBRANCH *)
TRACKER = (CPL == 3) ? IA32_U_CET.TRACKER : IA32_S_CET.TRACKER
IF EndbranchEnabledAndNotSuppressed(CPL) and TRACKER = WAIT_FOR_ENDBRANCH and
  (EAX != ERESUME or CR0.TS or (in SMM) or (CPUID.SGX_LEAF.0:EAX.SE1 = 0) or (CPL < 3))
    THEN
        Handle CET State machine violation      (* see Section 18.3.6, "Legacy Compatibility Treatment" in the
                                                   Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1. *)
FI;

IF CR0.PE= 0 or RFLAGS.VM = 1 or in SMM or CPUID.SGX_LEAF.0:EAX.SE1 = 0
    THEN #UD; FI;

IF CR0.TS = 1
    THEN #NM; FI;

IF CPL < 3
    THEN #UD; FI;

IF IA32_FEATURE_CONTROL.LOCK = 0 or IA32_FEATURE_CONTROL.SGX_ENABLE = 0
    THEN #GP(0); FI;
```

```
IF EAX is invalid leaf number
    THEN #GP(0); FI;

IF CR0.PG = 0 or CR0.NE = 0
    THEN #GP(0); FI;

IN_64BIT_MODE := IA32_EFER.LMA AND CS.L ? 1 : 0;
(* Check not in 16-bit mode and DS is not a 16-bit segment *)
IF not in 64-bit mode and CS.D = 0
    THEN #GP(0); FI;

IF CR_ENCLAVE_MODE = 1 and (EAX = 2 or EAX = 3) (* EENTER or ERESUME *)
    THEN #GP(0); FI;

IF CR_ENCLAVE_MODE = 0 and (EAX = 0 or EAX = 1 or EAX = 4 or EAX = 5 or EAX = 6 or EAX = 7)
(* EREPORT, EGETKEY, EEXIT, EACCEPT, EMODPE, or EACCEPTCOPY *)
    THEN #GP(0); FI;

Jump to leaf specific flow
```

Flags Affected

See individual leaf functions

Protected Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 3.
	If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.
	If input value in EAX encodes EENTER/ERESUME and ENCLAVE_MODE = 1.
	If input value in EAX encodes EGETKEY/EREPORT/EEXIT/EACCEPT/EACCEPTCOPY/EMODPE and ENCLAVE_MODE = 0.
	If operating in 16-bit mode.
	If data segment is in 16-bit mode.
	If CR0.PG = 0 or CR0.NE= 0.
#NM	If CR0.TS = 1.

Real-Address Mode Exceptions

#UD	ENCLS is not recognized in real mode.

Virtual-8086 Mode Exceptions

#UD	ENCLS is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 3.
	If CPUID.(EAX=12H,ECX=0):EAX.SGX1 [bit 0] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.
	If input value in EAX encodes EENTER/ERESUME and ENCLAVE_MODE = 1.
	If input value in EAX encodes EGETKEY/EREPORT/EEXIT/EACCEPT/EACCEPTCOPY/EMODPE and ENCLAVE_MODE = 0.
	If CR0.NE= 0.
#NM	If CR0.TS = 1.

ENCLV—Execute an Enclave VMM Function of Specified Leaf Number

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 C0 ENCLV	ZO	V/V	NA	This instruction is used to execute privileged SGX leaf functions that are reserved for VMM use. They are used for managing the enclaves.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Implicit Register Operands
ZO	NA	NA	NA	See Section 37.3

Description

The ENCLV instruction invokes the virtualization SGX leaf functions for managing enclaves in a virtualized environment. Software specifies the leaf function by setting the appropriate value in the register EAX as input. The registers RBX, RCX, and RDX have leaf-specific purpose, and may act as input, as output, or may be unused. In non 64-bit mode, the instruction ignores upper 32 bits of the RAX register.

The ENCLV instruction produces an invalid-opcode exception (#UD) if CR0.PE = 0 or RFLAGS.VM = 1, if it is executed in system-management mode (SMM), or not in VMX operation. Additionally, any attempt to execute the instruction when CPL > 0 results in #UD. The instruction produces a general-protection exception (#GP) if CR0.PG = 0 or if an attempt is made to invoke an undefined leaf function.

Software in VMX root mode of operation can enable execution of the ENCLV instruction in VMX non-root mode by setting enable ENCLV execution control in the VMCS. If enable ENCLV execution control in the VMCS is clear, execution of the ENCLV instruction in VMX non-root mode results in #UD.

When execution of ENCLV instruction in VMX non-root mode is enabled, software in VMX root operation can intercept the invocation of various ENCLV leaf functions in VMX non-root operation by setting the corresponding bits in the ENCLV-exiting bitmap.

Addresses and operands are 32 bits in 32-bit mode (IA32_EFER.LMA == 0 || CS.L == 0) and are 64 bits in 64-bit mode (IA32_EFER.LMA == 1 && CS.L == 1). CS.D value has no impact on address calculation.

Segment override prefixes and address-size override prefixes are ignored, as is the REX prefix in 64-bit mode.

Operation

```
IF TSX_ACTIVE
    THEN GOTO TSX_ABORT_PROCESSING; FI;

IF CR0.PE = 0 or RFLAGS.VM = 1 or in SMM or CPUID.SGX_LEAF.0:EAX.OSS = 0
    THEN #UD; FI;

IF not in VMX Operation or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD; FI;

IF (CPL > 0)
    THEN #UD; FI;

IF in VMX non-root operation
  IF "enable ENCLV exiting" VM-execution control is 1
    THEN
      IF EAX < 63 and ENCLV_exiting_bitmap[EAX] = 1 or EAX> 62 and ENCLV_exiting_bitmap[63] = 1
          THEN VM exit;
      FI;
  ELSE
    #UD; FI;
```

```
FI;

IF IA32_FEATURE_CONTROL.LOCK = 0 or IA32_FEATURE_CONTROL.SGX_ENABLE = 0
    THEN #GP(0); FI;

IF (EAX is an invalid leaf number)
    THEN #GP(0); FI;

IF CR0.PG = 0
    THEN #GP(0); FI;

(* DS must not be an expanded down segment *)
IF not in 64-bit mode and DS.Type is expand-down data
    THEN #GP(0); FI;

Jump to leaf specific flow
```

Flags Affected

See individual leaf functions.

Protected Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 0.
	If CPUID.(EAX=12H,ECX=0):EAX.OSS [bit 5] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.
	If data segment expand down.
	If CR0.PG=0.

Real-Address Mode Exceptions

#UD	ENCLV is not recognized in real mode.

Virtual-8086 Mode Exceptions

#UD	ENCLV is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#UD	If any of the LOCK/66H/REP/VEX prefixes are used.
	If current privilege level is not 0.
	If CPUID.(EAX=12H,ECX=0):EAX.OSS [bit 5] = 0.
	If logical processor is in SMM.
#GP(0)	If IA32_FEATURE_CONTROL.LOCK = 0.
	If IA32_FEATURE_CONTROL.SGX_ENABLE = 0.
	If input value in EAX encodes an unsupported leaf.

37.3 INTEL® SGX SYSTEM LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLS instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional implicit registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of each implicit register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EADD—Add a Page to an Uninitialized Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLS[EADD]	IR	V/V	SGX1	This leaf function adds a page to an uninitialized enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EADD (In)	Address of a PAGEINFO (In)	Address of the destination EPC page (In)

Description

This leaf function copies a source page from non-enclave memory into the EPC, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in EPCM. As part of the association, the enclave offset and the security attributes are measured and extended into the SECS.MRENCLAVE. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a PAGEINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of EADD leaf function.

EADD Memory Parameter Semantics

PAGEINFO	PAGEINFO.SECS	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Read access permitted by Non Enclave	Read access permitted by Non Enclave	Write access permitted by Enclave

The instruction faults if any of the following:

EADD Faulting Conditions

The operands are not properly aligned.	Unsupported security attributes are set.
Refers to an invalid SECS.	Reference is made to an SECS that is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page.
The EPC page is already valid.	If security attributes specifies a TCS and the source page specifies unsupported TCS values or fields.
The SECS has been initialized.	The specified enclave offset is outside of the enclave address space.

Concurrency Restrictions

Table 37-8. Base Concurrency Restrictions of EADD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EADD	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP	

Table 37-9. Additional Concurrency Restrictions of EADD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EADD	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGE-INFO.SECS	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EADD Operational Flow

Name	Type	Size (bits)	Description
TMP_SRCPGE	Effective Address	32/64	Effective address of the source page.
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the page to be added.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:TMP_SECINFO.
TMP_LINADDR	Unsigned Integer	64	Holds the linear address to be stored in the EPCM and used to calculate TMP_ENCLAVEOFFSET.
TMP_ENCLAVEOFFSET	Enclave Offset	64	The page displacement from the enclave base address.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
IF (DS:RBX is not 32Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECS := DS:RBX.SECS;
TMP_SECINFO := DS:RBX.SECINFO;
TMP_LINADDR := DS:RBX.LINADDR;

IF (DS:TMP_SRCPGE is not 4KByte aligned or DS:TMP_SECS is not 4KByte aligned or
    DS:TMP_SECINFO is not 64Byte aligned or TMP_LINADDR is not 4KByte aligned)
    THEN #GP(0); FI;

IF (DS:TMP_SECS does not resolve within an EPC)
    THEN #PF(DS:TMP_SECS); FI;

SCRATCH_SECINFO := DS:TMP_SECINFO;

(* Check for misconfigured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero or
```

```
    ! (SCRATCH_SECINFO.FLAGS.PT is PT_REG or SCRATCH_SECINFO.FLAGS.PT is PT_TCS or
    (SCRATCH_SECINFO.FLAGS.PT is PT_SS_FIRST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1) or
    (SCRATCH_SECINFO.FLAGS.PT is PT_SS_REST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1)) )
    THEN #GP(0); FI;

(* If PT_SS_FIRST/PT_SS_REST page types are requested then CR4.CET must be 1 *)
IF ( (SCRATCH_SECINFO.FLAGS.PT is PT_SS_FIRST OR
    SCRATCH_SECINFO.FLAGS.PT is PT_SS_REST) AND CR4.CET == 0)
    THEN #GP(0); FI;

(* Check the EPC page for concurrency *)
IF (EPC page is not available for EADD)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
                VMCS.Guest-linear_address := DS:RCX;
            Deliver VMEXIT;
            ELSE
                #GP(0);
        FI;
FI;

IF (EPCM(DS:RCX).VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

(* Check the SECS for concurrency *)
IF (SECS is not available for EADD)
    THEN #GP(0); FI;

IF (EPCM(DS:TMP_SECS).VALID = 0 or EPCM(DS:TMP_SECS).PT ≠ PT_SECS)
    THEN #PF(DS:TMP_SECS); FI;

(* Copy 4KBytes from source page to EPC page*)
DS:RCX[32767:0] := DS:TMP_SRCPGE[32767:0];

CASE (SCRATCH_SECINFO.FLAGS.PT)

    PT_TCS:
        IF (DS:RCX.RESERVED ≠ 0) #GP(0); FI;
        IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and
            ((DS:TCS.FSLIMIT & 0FFFH ≠ 0FFFH) or (DS:TCS.GSLIMIT & 0FFFH ≠ 0FFFH) )) #GP(0); FI;
        (* Ensure TCS.PREVSSP is zero *)
        IF (CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1) and (DS:RCX.PREVSSP != 0) #GP(0); FI;
        BREAK;
    PT_REG:
        IF (SCRATCH_SECINFO.FLAGS.W = 1 and SCRATCH_SECINFO.FLAGS.R = 0) #GP(0); FI;
        BREAK;
    PT_SS_FIRST:
    PT_SS_REST:
    (* SS pages cannot created on first or last page of ELRANGE *)
```

```
    IF ( TMP_LINADDR = DS:TMP_SECS.BASEADDR or TMP_LINADDR = (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE - 0x1000) )
        THEN #GP(0); FI;
    IF ( DS:RCX[4087:0] != 0 ) #GP(0); FI;
    IF (SCRATCH_SECINFO.FLAGS.PT == PT_SS_FIRST)
        THEN
            (* Check that valid RSTORSSP token exists *)
            IF ( DS:RCX[4095:4088] != ((TMP_LINADDR + 0x1000) | DS:TMP_SECS.ATTRIBUTES.MODE64BIT) ) #GP(0); FI;
            (* Check the 8 bytes are zero *)
            IF ( DS:RCX[4095:4088] != 0 ) #GP(0); FI;
    FI;
    IF (SCRATCH_SECINFO.FLAGS.W = 0 OR SCRATCH_SECINFO.FLAGS.R = 0 OR
     SCRATCH_SECINFO.FLAGS.X = 1) #GP(0); FI;
        BREAK;
ESAC;

(* Check the enclave offset is within the enclave linear address space *)
IF (TMP_LINADDR < DS:TMP_SECS.BASEADDR or TMP_LINADDR ≥ DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE)
    THEN #GP(0); FI;

(* Check concurrency of measurement resource*)
IF (Measurement being updated)
    THEN #GP(0); FI;

(* Check if the enclave to which the page will be added is already in Initialized state *)
IF (DS:TMP_SECS already initialized)
    THEN #GP(0); FI;

(* For TCS pages, force EPCM.rwx bits to 0 and no debug access *)
IF (SCRATCH_SECINFO.FLAGS.PT = PT_TCS)
    THEN
        SCRATCH_SECINFO.FLAGS.R := 0;
        SCRATCH_SECINFO.FLAGS.W := 0;
        SCRATCH_SECINFO.FLAGS.X := 0;
        (DS:RCX).FLAGS.DBGOPTIN := 0; // force TCS.FLAGS.DBGOPTIN off
        DS:RCX.CSSA := 0;
        DS:RCX.AEP := 0;
        DS:RCX.STATE := 0;
FI;

(* Add enclave offset and security attributes to MRENCLAVE *)
TMP_ENCLAVEOFFSET := TMP_LINADDR - DS:TMP_SECS.BASEADDR;
TMPUPDATEFIELD[63:0] := 0000000044444145H; // "EADD"
TMPUPDATEFIELD[127:64] := TMP_ENCLAVEOFFSET;
TMPUPDATEFIELD[511:128] := SCRATCH_SECINFO[375:0]; // 48 bytes
DS:TMP_SECS.MRENCLAVE := SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

(* Add enclave offset and security attributes to MRENCLAVE *)
EPCM(DS:RCX).R := SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PT := SCRATCH_SECINFO.FLAGS.PT;
EPCM(DS:RCX).ENCLAVEADDRESS := TMP_LINADDR;
```

```
(* associate the EPCPAGE with the SECS by storing the SECS identifier of DS:TMP_SECS *)
Update EPCM(DS:RCX) SECS identifier to reference DS:TMP_SECS identifier;

(* Set EPCM entry fields *)
EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).VALID := 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If an enclave memory operand is outside of the EPC.
	If an enclave memory operand is the wrong type.
	If a memory operand is locked.
	If the enclave is initialized.
	If the enclave's MRENCLAVE is locked.
	If the TCS page reserved bits are set.
	If the TCS page PREVSSP field is not zero.
	If the PT_SS_REST or PT_SS_REST page is the first or last page in the enclave.
	If the PT_SS_FIRST or PT_SS_REST page is not initialized correctly.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the EPC page is valid.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If an enclave memory operand is outside of the EPC.
	If an enclave memory operand is the wrong type.
	If a memory operand is locked.
	If the enclave is initialized.
	If the enclave's MRENCLAVE is locked.
	If the TCS page reserved bits are set.
	If the TCS page PREVSSP field is not zero.
	If the PT_SS_REST or PT_SS_REST page is the first or last page in the enclave.
	If the PT_SS_FIRST or PT_SS_REST page is not initialized correctly.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the EPC page is valid.

EAUG—Add a Page to an Initialized Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0DH ENCLS[EAUG]	IR	V/V	SGX2	This leaf function adds a page to an initialized enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EAUG (In)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function zeroes a page of EPC memory, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in the EPCM. As part of the association, the security attributes are configured to prevent access to the EPC page until a corresponding invocation of the EACCEPT leaf or EACCEPT-COPY leaf confirms the addition of the new page into the enclave. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a PAGEINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EAUG leaf function.

EAUG Memory Parameter Semantics

PAGEINFO	PAGEINFO.SECS	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Must be zero	Read access permitted by Non Enclave	Write access permitted by Enclave

The instruction faults if any of the following:

EAUG Faulting Conditions

The operands are not properly aligned.	Unsupported security attributes are set.
Refers to an invalid SECS.	Reference is made to an SECS that is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page.
The EPC page is already valid.	The specified enclave offset is outside of the enclave address space.
The SECS has been initialized.	

Concurrency Restrictions

Table 37-10. Base Concurrency Restrictions of EAUG

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EAUG	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP	

Table 37-11. Additional Concurrency Restrictions of EAUG

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EAUG	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGE-INFO.SECS	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EAUG Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the page to be added.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:TMP_SECINFO.
TMP_LINADDR	Unsigned Integer	64	Holds the linear address to be stored in the EPCM and used to calculate TMP_ENCLAVEOFFSET.

```
IF (DS:RBX is not 32Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SECS := DS:RBX.SECS;
TMP_SECINFO := DS:RBX.SECINFO;
IF (DS:RBX.SECINFO is not 0)
    THEN
        IF (DS:TMP_SECINFO is not 64B aligned)
            THEN #GP(0); FI;
FI;

TMP_LINADDR := DS:RBX.LINADDR;

IF ( DS:TMP_SECS is not 4KByte aligned or TMP_LINADDR is not 4KByte aligned )
    THEN #GP(0); FI;

IF DS:RBX.SRCPAGE is not 0
    THEN #GP(0); FI;

IF (DS:TMP_SECS does not resolve within an EPC)
    THEN #PF(DS:TMP_SECS); FI;

(* Check the EPC page for concurrency *)
```

```
IF (EPC page in use)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
                VMCS.Guest-linear_address := DS:RCX;
                Deliver VMEXIT;
            ELSE
                #GP(0);
        FI;
FI:

IF (EPCM(DS:RCX).VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

(* copy SECINFO contents into a scratch SECINFO *)
IF (DS:RBX.SECINFO is 0)
    THEN
        (* allocate and initialize a new scratch SECINFO structure *)
        SCRATCH_SECINFO.PT := PT_REG;
        SCRATCH_SECINFO.R := 1;
        SCRATCH_SECINFO.W := 1;
        SCRATCH_SECINFO.X := 0;
        << zero out remaining fields of SCRATCH_SECINFO >>
    ELSE
        (* copy SECINFO contents into scratch SECINFO *)
        SCRATCH_SECINFO := DS:TMP_SECINFO;
        (* check SECINFO flags for misconfiguration *)
        (* reserved flags must be zero *)
        (* SECINFO.FLAGS.PT must either be PT_SS_FIRST, or PT_SS_REST *)
        IF ( (SCRATCH_SECINFO reserved fields are not 0) or
        CPUID.(EAX=12H, ECX=1):EAX[6] is 0) OR
         (SCRATCH_SECINFO.PT is not PT_SS_FIRST, or PT_SS_REST) OR
         ( (SCRATCH_SECINFO.FLAGS.R is 0) OR (SCRATCH_SECINFO.FLAGS.W is 0) OR (SCRATCH_SECINFO.FLAGS.X is 1) ) )
            THEN #GP(0); FI;
FI;
(* Check if PT_SS_FIRST/PT_SS_REST page types are requested then CR4.CET must be 1 *)
IF ( (SCRATCH_SECINFO.PT is PT_SS_FIRST OR SCRATCH_SECINFO.PT is PT_SS_REST) AND CR4.CET == 0 )
    THEN #GP(0); FI;

(* Check the SECS for concurrency *)
IF (SECS is not available for EAUG)
    THEN #GP(0); FI;

IF (EPCM(DS:TMP_SECS).VALID = 0 or EPCM(DS:TMP_SECS).PT ≠ PT_SECS)
    THEN #PF(DS:TMP_SECS); FI;

(* Check if the enclave to which the page will be added is in the Initialized state *)
IF (DS:TMP_SECS is not initialized)
    THEN #GP(0); FI;
```

```
(* Check the enclave offset is within the enclave linear address space *)
IF ( (TMP_LINADDR < DS:TMP_SECS.BASEADDR) or (TMP_LINADDR ≥ DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE) )
    THEN #GP(0); FI;

IF ( (SCRATCH_SECINFO.PT is PT_SS_FIRST OR SCRATCH_SECINFO.PT is PT_SS_REST) )
    THEN
        (* SS pages cannot created on first or last page of ELRANGE *)
        IF ( TMP_LINADDR == DS:TMP_SECS.BASEADDR OR
         TMP_LINADDR == (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE - 0x1000) )
            THEN
                #GP(0); FI;
FI;

(* Clear the content of EPC page*)
DS:RCX[32767:0] := 0;

IF (CPUID.(EAX=07H, ECX=0H):ECX[CET_SS] = 1)
    THEN
        (* set up shadow stack RSTORSSP token *)
        IF (SCRATCH_SECINFO.PT is PT_SS_FIRST)
        THEN
            DS:RCX[0xFF8] := (TMP_LINADDR + 0x1000) | TMP_SECS.ATTRIBUTES.MODE64BIT; FI;
FI;

(* Set EPCM security attributes *)
EPCM(DS:RCX).R := SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PT := SCRATCH_SECINFO.FLAGS.PT;
EPCM(DS:RCX).ENCLAVEADDRESS := TMP_LINADDR;
EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 1;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;

(* associate the EPCPAGE with the SECS by storing the SECS identifier of DS:TMP_SECS *)
Update EPCM(DS:RCX) SECS identifier to reference DS:TMP_SECS identifier;

(* Set EPCM valid fields *)
EPCM(DS:RCX).VALID := 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit. If a memory operand is not properly aligned. If a memory operand is locked. If the enclave is not initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form. If a memory operand is not properly aligned. If a memory operand is locked. If the enclave is not initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.

EBLOCK—Mark a page in EPC as Blocked

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 09H ENCLS[EBLOCK]	IR	V/V	SGX1	This leaf function marks a page in the EPC as blocked.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	EBLOCK (In)	Return error code (Out)	Effective address of the EPC page (In)

Description

This leaf function causes an EPC page to be marked as BLOCKED. This instruction can only be executed when current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create linear address. Segment override is not supported.

An error code is returned in RAX.

The table below provides additional information on the memory parameter of EBLOCK leaf function.

EBLOCK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 37-12. EBLOCK Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	EBLOCK successful.
SGX_BLKSTATE	Page already blocked. This value is used to indicate to a VMM that the page was already in BLOCKED state as a result of EBLOCK and thus will need to be restored to this state when it is eventually reloaded (using ELDB).
SGX_ENTRYEPOCH_LOCKED	SECS locked for Entry Epoch update. This value indicates that an ETRACK is currently executing on the SECS. The EBLOCK should be reattempted.
SGX_NOTBLOCKABLE	Page type is not one which can be blocked.
SGX_PG_INVLD	Page is not valid and cannot be blocked.
SGX_EPC_PAGE_CONFLICT	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODT, or EWB.

Concurrency Restrictions

Table 37-13. Base Concurrency Restrictions of EBLOCK

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EBLOCK	Target [DS:RCX]	Shared	SGX_EPC_PAGE_ CONFLICT	

Table 37-14. Additional Concurrency Restrictions of EBLOCK

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EBLOCK	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EBLOCK Operational Flow

Name	Type	Size (Bits)	Description
TMP_BLKSTATE	Integer	64	Page is already blocked.

```
IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

RFLAGS.ZF,CF,PF,AF,OF,SF := 0;
RAX := 0;

(* Check the EPC page for concurrency*)
IF (EPC page in use)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_EPC_PAGE_CONFLICT;
        GOTO DONE;
FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_PG_INVLD;
        GOTO DONE;
FI;

IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS) and (EPCM(DS:RCX).PT ≠ PT_TRIM)
 and EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
    THEN
        RFLAGS.CF := 1;
        IF (EPCM(DS:RCX).PT = PT_SECS)
            THEN RAX := SGX_PG_IS_SECS;
            ELSE RAX := SGX_NOTBLOCKABLE;
        FI;
        GOTO DONE;
FI;

(* Check if the page is already blocked and report blocked state *)
TMP_BLKSTATE := EPCM(DS:RCX).BLOCKED;
```

```
(* at this point, the page must be valid and PT_TCS or PT_REG or PT_TRIM*)
IF (TMP_BLKSTATE = 1)
    THEN
        RFLAGS.CF := 1;
        RAX := SGX_BLKSTATE;
    ELSE
        EPCM(DS:RCX).BLOCKED := 1
FI;
DONE:
```

Flags Affected

Sets ZF if SECS is in use or invalid, otherwise cleared. Sets CF if page is BLOCKED or not blockable, otherwise cleared. Clears PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the specified EPC resource is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If the specified EPC resource is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

ECREATE—Create an SECS page in the Enclave Page Cache

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLS[ECREATE]	IR	V/V	SGX1	This leaf function begins an enclave build by creating an SECS page in EPC.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	ECREATE (In)	Address of a PAGEINFO (In)	Address of the destination SECS page (In)

Description

ENCLS[ECREATE] is the first instruction executed in the enclave build process. ECREATE copies an SECS structure outside the EPC into an SECS page inside the EPC. The internal structure of SECS is not accessible to software.

ECREATE will set up fields in the protected SECS and mark the page as valid inside the EPC. ECREATE initializes or checks unused fields.

Software sets the following fields in the source structure: SECS:BASEADDR, SECS:SIZE in bytes, ATTRIBUTES, CONFIGID and CONFIGSVN. SECS:BASEADDR must be naturally aligned on an SECS.SIZE boundary. SECS.SIZE must be at least 2 pages (8192).

The source operand RBX contains an effective address of a PAGEINFO structure. PAGEINFO contains an effective address of a source SECS and an effective address of an SECINFO. The SECS field in PAGEINFO is not used.

The RCX register is the effective address of the destination SECS. It is an address of an empty slot in the EPC. The SECS structure must be page aligned. SECINFO flags must specify the page as an SECS page.

ECREATE Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read access permitted by Non Enclave	Read access permitted by Non Enclave	Write access permitted by Enclave

ECREATE will fault if the SECS target page is in use; already valid; outside the EPC. It will also fault if addresses are not aligned; unused PAGEINFO fields are not zero.

If the amount of space needed to store the SSA frame is greater than the amount specified in SECS.SSAFRAME-SIZE, a #GP(0) results. The amount of space needed for an SSA frame is computed based on DS:TMP_SECS.ATTRIBUTES.XFRM size. Details of computing the size can be found Section 38.7.

Concurrency Restrictions

Table 37-15. Base Concurrency Restrictions of ECREATE

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ECREATE	SECS [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION

Table 37-16. Additional Concurrency Restrictions of ECREATE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ECREATE	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ECREATE Operational Flow

Name	Type	Size (Bits)	Description
TMP_SRCPGE	Effective Address	32/64	Effective address of the SECS source page.
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the SECS page to be added.
TMP_XSIZE	SSA Size	64	The size calculation of SSA frame.
TMP_MISC_SIZE	MISC Field Size	64	Size of the selected MISC field components.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
IF (DS:RBX is not 32Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECINFO := DS:RBX.SECINFO;

IF (DS:TMP_SRCPGE is not 4KByte aligned or DS:TMP_SECINFO is not 64Byte aligned)
    THEN #GP(0); FI;

IF (DS:RBX.LINADDR ! = 0 or DS:RBX.SECS ≠ 0)
    THEN #GP(0); FI;

(* Check for misconfigured SECINFO flags*)
IF (DS:TMP_SECINFO reserved fields are not zero or DS:TMP_SECINFO.FLAGS.PT ≠ PT_SECS)
    THEN #GP(0); FI;

TMP_SECS := RCX;

IF (EPC entry in use)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
```

```
                    VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                    VMCS.Exit_qualification.error := 0;
                    VMCS.Guest-physical_address :=
                        << translation of DS:TMP_SECS produced by paging >>;
                    VMCS.Guest-linear_address := DS:TMP_SECS;
                Deliver VMEXIT;
                ELSE
                    #GP(0);
            FI;
FI;

IF (EPC entry in use)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID = 1)
    THEN #PF(DS:RCX); FI;

(* Copy 4KBytes from source page to EPC page*)
DS:RCX[32767:0] := DS:TMP_SRCPGE[32767:0];

(* Check lower 2 bits of XFRM are set *)
IF ( ( DS:TMP_SECS.ATTRIBUTES.XFRM BitwiseAND 03H) ≠ 03H)
    THEN #GP(0); FI;

IF (XFRM is illegal)
    THEN #GP(0); FI;

(* Check legality of CET_ATTRIBUTES *)
IF ((DS:TMP_SECS.ATTRIBUTES.CET = 0 and DS:TMP_SECS.CET_ATTRIBUTES ≠ 0) ||
    (DS:TMP_SECS.ATTRIBUTES.CET = 0 and DS:TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):EDX[CET_IBT] = 0 and DS:TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):EDX[CET_IBT] = 0 and DS:TMP_SECS.CET_ATTRIBUTES[5:2] ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):ECX[CET_SS] = 0 and DS:TMP_SECS.CET_ATTRIBUTES[1:0] ≠ 0) ||
    (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1 and
     (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) not canonical) ||
    (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0 and
     (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) & 0xFFFFFFFF00000000) ||
    (DS:TMP_SECS.CET_ATTRIBUTES.reserved fields not 0) or
     (DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) is not page aligned))
    THEN
        #GP(0);
FI;

(* Make sure that the SECS does not have any unsupported MISCSELECT options*)
IF ( !(CPUID.(EAX=12H, ECX=0):EBX[31:0] & DS:TMP_SECS.MISCSELECT[31:0]) )
    THEN
        EPCM(DS:TMP_SECS).EntryLock.Release();
        #GP(0);
FI;

( * Compute size of MISC area *)
TMP_MISC_SIZE := compute_misc_region_size();

(* Compute the size required to save state of the enclave on async exit, see Section 38.7.2.2*)
```

```
TMP_XSIZE := compute_xsave_size(DS:TMP_SECS.ATTRIBUTES.XFRM) + GPR_SIZE + TMP_MISC_SIZE;

(* Ensure that the declared area is large enough to hold XSAVE and GPR stat *)
IF ( DS:TMP_SECS.SSAFRAMESIZE*4096 < TMP_XSIZE)
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.BASEADDR is not canonical) )
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and (DS:TMP_SECS.BASEADDR and 0FFFFFFFF00000000H) )
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and (DS:TMP_SECS.SIZE ≥ 2 ^ (CPUID.(EAX=12H, ECX=0):.EDX[7:0]) ) )
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.SIZE ≥ 2 ^ (CPUID.(EAX=12H, ECX=0):.EDX[15:8]) ) )
    THEN #GP(0); FI;

(* Enclave size must be at least 8192 bytes and must be power of 2 in bytes*)
IF (DS:TMP_SECS.SIZE < 8192 or popcnt(DS:TMP_SECS.SIZE) > 1)
    THEN #GP(0); FI;

(* Ensure base address of an enclave is aligned on size*)
IF ( ( DS:TMP_SECS.BASEADDR and (DS:TMP_SECS.SIZE-1) ) )
    THEN #GP(0); FI;

(* Ensure the SECS does not have any unsupported attributes*)
IF ( DS:TMP_SECS.ATTRIBUTES and (~CR_SGX_ATTRIBUTES_MASK) )
    THEN #GP(0); FI;

IF ( DS:TMP_SECS reserved fields are not zero)
    THEN #GP(0); FI;

(* Verify that CONFIGID/CONFIGSVN are not set with attribute *)
IF ( ((DS:TMP_SECS.CONFIGID ≠ 0) or (DS:TMP_SECS.CONFIGSVN ≠0)) AND (DS:TMP_SECS.ATTRIBUTES.KSS == 0 ))
    THEN #GP(0); FI;

Clear DS:TMP_SECS to Uninitialized;
DS:TMP_SECS.MRENCLAVE := SHA256INITIALIZE(DS:TMP_SECS.MRENCLAVE);
DS:TMP_SECS.ISVSVN := 0;
DS:TMP_SECS.ISVPRODID := 0;

(* Initialize hash updates etc*)
Initialize enclave's MRENCLAVE update counter;

(* Add "ECREATE" string and SECS fields to MRENCLAVE *)
TMPUPDATEFIELD[63:0] := 0045544145524345H; // "ECREATE"
TMPUPDATEFIELD[95:64] := DS:TMP_SECS.SSAFRAMESIZE;
TMPUPDATEFIELD[159:96] := DS:TMP_SECS.SIZE;
IF (CPUID.(EAX=7, ECX=0):EDX[CET_IBT] = 1)
    THEN
        TMPUPDATEFIELD[223:160] := DS:TMP_SECS.CET_LEG_BITMAP_OFFSET;
    ELSE
        TMPUPDATEFIELD[223:160] := 0;
```

```
FI;
TMPUPDATEFIELD[511:160] := 0;
DS:TMP_SECS.MRENCLAVE := SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

(* Set EID *)
DS:TMP_SECS.EID := LockedXAdd(CR_NEXT_EID, 1);

(* Initialize the virtual child count to zero *)
DS:TMP_SECS.VIRTCHILDCNT := 0;

(* Load ENCLAVECONTEXT with Address out of paging of SECS *)
<< store translation of DS:RCX produced by paging in SECS(DS:RCX).ENCLAVECONTEXT >>

(* Set the EPCM entry, first create SECS identifier and store the identifier in EPCM *)
EPCM(DS:TMP_SECS).PT := PT_SECS;
EPCM(DS:TMP_SECS).ENCLAVEADDRESS := 0;
EPCM(DS:TMP_SECS).R := 0;
EPCM(DS:TMP_SECS).W := 0;
EPCM(DS:TMP_SECS).X := 0;

(* Set EPCM entry fields *)
EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;
EPCM(DS:RCX).VALID := 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the reserved fields are not zero.
	If PAGEINFO.SECS is not zero.
	If PAGEINFO.LINADDR is not zero.
	If the SECS destination is locked.
	If SECS.SSAFRAMESIZE is insufficient.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the SECS destination is outside the EPC.

64-Bit Mode Exceptions

#GP(0)	If a memory address is non-canonical form.
	If a memory operand is not properly aligned.
	If the reserved fields are not zero.
	If PAGEINFO.SECS is not zero.
	If PAGEINFO.LINADDR is not zero.
	If the SECS destination is locked.
	If SECS.SSAFRAMESIZE is insufficient.

#PF(error code)	If a page fault occurs in accessing memory operands. If the SECS destination is outside the EPC.

EDBGRD—Read From a Debug Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 04H ENCLS[EDBGRD]	IR	V/V	SGX1	This leaf function reads a dword/quadword from a debug enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EDBGRD (In)	Return error code (Out)	Data read from a debug enclave (Out)	Address of source memory in the EPC (In)

Description

This leaf function copies a quadword/doubleword from an EPC page belonging to a debug enclave into the RBX register. Eight bytes are read in 64-bit mode, four bytes are read in non-64-bit modes. The size of data read cannot be overridden.

The effective address of the source location inside the EPC is provided in the register RCX.

EDBGRD Memory Parameter Semantics

EPCQW
Read access permitted by Enclave

The error codes are:

Table 37-17. EDBGRD Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	EDBGRD successful.
SGX_PAGE_NOT_DEBUGGABLE	The EPC page cannot be accessed because it is in the PENDING or MODIFIED state.

The instruction faults if any of the following:

EDBGRD Faulting Conditions

RCX points into a page that is an SECS.	RCX does not resolve to a naturally aligned linear address.
RCX points to a page that does not belong to an enclave that is in debug mode.	RCX points to a location inside a TCS that is beyond the architectural size of the TCS (SGX_TCS_LIMIT).
An operand causing any segment violation.	May page fault.
CPL > 0.	

This instruction ignores the EPCM RWX attributes on the enclave page. Consequently, violation of EPCM RWX attributes via EDBGRD does not result in a #GP.

Concurrency Restrictions

Table 37-18. Base Concurrency Restrictions of EDBGRD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDBGRD	Target [DS:RCX]	Shared	#GP	

Table 37-19. Additional Concurrency Restrictions of EDBGRD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDBGRD	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDBGRD Operational Flow

Name	Type	Size (Bits)	Description
TMP_MODE64	Binary	1	((IA32_EFER.LMA = 1) && (CS.L = 1))
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF ( (TMP_MODE64 = 1) and (DS:RCX is not 8Byte Aligned) )
    THEN #GP(0); FI;

IF ( (TMP_MODE64 = 0) and (DS:RCX is not 4Byte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* make sure no other Intel SGX instruction is accessing the same EPCM entry *)
IF (Another instruction modifying the same EPCM entry is executing)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (SOURCE) is pointing to a PT_REG or PT_TCS or PT_VA or PT_SS_FIRST or PT_SS_REST *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS) and (EPCM(DS:RCX).PT ≠ PT_VA)
and (EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX points to an accessible EPC page *)
IF (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0) )
    THEN
        RFLAGS.ZF := 1;
```

```
        RAX := SGX_PAGE_NOT_DEBUGGABLE;
        GOTO DONE;
FI;

(* If source is a TCS, then make sure that the offset into the page is not beyond the TCS size*)
IF ( ( EPCM(DS:RCX). PT = PT_TCS) and ((DS:RCX) & FFFH ≥ SGX_TCS_LIMIT) )
    THEN #GP(0); FI;

(* make sure the enclave owning the PT_REG or PT_TCS page allow debug *)
IF ( (EPCM(DS:RCX).PT = PT_REG) or (EPCM(DS:RCX).PT = PT_TCS) )
    THEN
        TMP_SECS := GET_SECS_ADDRESS;
        IF (TMP_SECS.ATTRIBUTES.DEBUG = 0)
            THEN #GP(0); FI;
        IF ( (TMP_MODE64 = 1) )
            THEN RBX[63:0] := (DS:RCX)[63:0];
            ELSE EBX[31:0] := (DS:RCX)[31:0];
        FI;
    ELSE
        TMP_64BIT_VAL[63:0] := (DS:RCX)[63:0] & (~07H); // Read contents from VA slot
        IF (TMP_MODE64 = 1)
            THEN
                IF (TMP_64BIT_VAL ≠ 0H)
                    THEN RBX[63:0] := 0FFFFFFFFFFFFFFFFH;
                    ELSE RBX[63:0] := 0H;
                FI;
            ELSE
                IF (TMP_64BIT_VAL ≠ 0H)
                    THEN EBX[31:0] := 0FFFFFFFFH;
                    ELSE EBX[31:0] := 0H;
                FI;
FI;

(* clear EAX and ZF to indicate successful completion *)
RAX := 0;
RFLAGS.ZF := 0;

DONE:
(* clear flags *)
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

ZF is set if the page is MODIFIED or PENDING; RAX contains the error code. Otherwise ZF is cleared and RAX is set to 0. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If the address in RCS violates DS limit or access rights.
	If DS segment is unusable.
	If RCX points to a memory location not 4Byte-aligned.
	If the address in RCX points to a page belonging to a non-debug enclave.
	If the address in RCX points to a page which is not PT_TCS, PT_REG or PT_VA.
	If the address in RCX points to a location inside TCS that is beyond SGX_TCS_LIMIT.

#PF(error code)	If a page fault occurs in accessing memory operands. If the address in RCX points to a non-EPC page. If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form. If RCX points to a memory location not 8Byte-aligned. If the address in RCX points to a page belonging to a non-debug enclave. If the address in RCX points to a page which is not PT_TCS, PT_REG or PT_VA. If the address in RCX points to a location inside TCS that is beyond SGX_TCS_LIMIT.
#PF(error code)	If a page fault occurs in accessing memory operands. If the address in RCX points to a non-EPC page. If the address in RCX points to an invalid EPC page.

EDBGWR—Write to a Debug Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 05H ENCLS[EDBGWR]	IR	V/V	SGX1	This leaf function writes a dword/quadword to a debug enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EDBGWR (In)	Return error code (Out)	Data to be written to a debug enclave (In)	Address of Target memory in the EPC (In)

Description

This leaf function copies the content in EBX/RBX to an EPC page belonging to a debug enclave. Eight bytes are written in 64-bit mode, four bytes are written in non-64-bit modes. The size of data cannot be overridden.

The effective address of the target location inside the EPC is provided in the register RCX.

EDBGWR Memory Parameter Semantics

EPCQW
Write access permitted by Enclave

The instruction faults if any of the following:

EDBGWR Faulting Conditions

RCX points into a page that is an SECS.	RCX does not resolve to a naturally aligned linear address.
RCX points to a page that does not belong to an enclave that is in debug mode.	RCX points to a location inside a TCS that is not the FLAGS word.
An operand causing any segment violation.	May page fault.
CPL > 0.	

The error codes are:

Table 37-20. EDBGWR Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	EDBGWR successful.
SGX_PAGE_NOT_DEBUGGABLE	The EPC page cannot be accessed because it is in the PENDING or MODIFIED state.

This instruction ignores the EPCM RWX attributes on the enclave page. Consequently, violation of EPCM RWX attributes via EDBGRD does not result in a #GP.

Concurrency Restrictions

Table 37-21. Base Concurrency Restrictions of EDBGWR

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDBGWR	Target [DS:RCX]	Shared	#GP	

Table 37-22. Additional Concurrency Restrictions of EDBGWR

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDBGWR	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDBGWR Operational Flow

Name	Type	Size (Bits)	Description
TMP_MODE64	Binary	1	((IA32_EFER.LMA = 1) && (CS.L = 1)).
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF ( (TMP_MODE64 = 1) and (DS:RCX is not 8Byte Aligned) )
    THEN #GP(0); FI;

IF ( (TMP_MODE64 = 0) and (DS:RCX is not 4Byte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* make sure no other Intel SGX instruction is accessing the same EPCM entry *)
IF (Another instruction modifying the same EPCM entry is executing)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (DST) is pointing to a PT_REG or PT_TCS or PT_SS_FIRST or PT_SS_REST *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS)
  and (EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX points to an accessible EPC page *)
IF ( (EPCM(DS:RCX).PENDING is not 0) or (EPCM(DS:RCS).MODIFIED is not 0) )
    THEN
        RFLAGS.ZF := 1;
```

```
        RAX := SGX_PAGE_NOT_DEBUGGABLE;
        GOTO DONE;
FI;

(* If destination is a TCS, then make sure that the offset into the page can only point to the FLAGS field*)
IF ( ( EPCM(DS:RCX). PT = PT_TCS) and ((DS:RCX) & FF8H ≠ offset_of_FLAGS & 0FF8H) )
    THEN #GP(0); FI;

(* Locate the SECS for the enclave to which the DS:RCX page belongs *)
TMP_SECS := GET_SECS_PHYS_ADDRESS(EPCM(DS:RCX).ENCLAVESECS);

(* make sure the enclave owning the PT_REG or PT_TCS page allow debug *)
IF (TMP_SECS.ATTRIBUTES.DEBUG = 0)
    THEN #GP(0); FI;

IF ( (TMP_MODE64 = 1) )
    THEN (DS:RCX)[63:0] := RBX[63:0];
    ELSE (DS:RCX)[31:0] := EBX[31:0];
FI;

(* clear EAX and ZF to indicate successful completion *)
RAX := 0;
RFLAGS.ZF := 0;

DONE:
(* clear flags *)
RFLAGS.CF,PF,AF,OF,SF := 0
```

Flags Affected

ZF is set if the page is MODIFIED or PENDING; RAX contains the error code. Otherwise ZF is cleared and RAX is set to 0. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If the address in RCS violates DS limit or access rights.
	If DS segment is unusable.
	If RCX points to a memory location not 4Byte-aligned.
	If the address in RCX points to a page belonging to a non-debug enclave.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a location inside TCS that is not the FLAGS word.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RCX points to a non-EPC page.
	If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0)	If RCX is non-canonical form.
	If RCX points to a memory location not 8Byte-aligned.
	If the address in RCX points to a page belonging to a non-debug enclave.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a location inside TCS that is not the FLAGS word.

#PF(error code)	If a page fault occurs in accessing memory operands. If the address in RCX points to a non-EPC page. If the address in RCX points to an invalid EPC page.

EEXTEND—Extend Uninitialized Enclave Measurement by 256 Bytes

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 06H ENCLS[EEXTEND]	IR	V/V	SGX1	This leaf function measures 256 bytes of an uninitialized enclave page.

Instruction Operand Encoding

Op/En	EAX	EBX	RCX
IR	EEXTEND (In)	Effective address of the SECS of the data chunk (In)	Effective address of a 256-byte chunk in the EPC (In)

Description

This leaf function updates the MRENCLAVE measurement register of an SECS with the measurement of an EXTEND string compromising of "EEXTEND" || ENCLAVEOFFSET || PADDING || 256 bytes of the enclave page. This instruction can only be executed when current privilege level is 0 and the enclave is uninitialized.

RBX contains the effective address of the SECS of the region to be measured. The address must be the same as the one used to add the page into the enclave.

RCX contains the effective address of the 256 byte region of an EPC page to be measured. The DS segment is used to create linear addresses. Segment override is not supported.

EEXTEND Memory Parameter Semantics

EPC[RCX]
Read access by Enclave

The instruction faults if any of the following:

EEXTEND Faulting Conditions

RBX points to an address not 4KBytes aligned.	RBX does not resolve to an SECS.
RBX does not point to an SECS page.	RBX does not point to the SECS page of the data chunk.
RCX points to an address not 256B aligned.	RCX points to an unused page or a SECS.
RCX does not resolve in an EPC page.	If SECS is locked.
If the SECS is already initialized.	May page fault.
CPL > 0.	

Concurrency Restrictions

Table 37-23. Base Concurrency Restrictions of EEXTEND

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EEXTEND	Target [DS:RCX]	Shared	#GP	
	SECS [DS:RBX]	Concurrent		

Table 37-24. Additional Concurrency Restrictions of EEXTEND

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EEXTEND	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EEXTEND Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.
TMP_ENCLAVEOFFSET	Enclave Offset	64	The page displacement from the enclave base address.
TMPUPDATEFIELD	SHA256 Buffer	512	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF (DS:RBX is not 4096 Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve to an EPC page)
    THEN #PF(DS:RBX); FI;

IF (DS:RCX is not 256Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* make sure no other Intel SGX instruction is accessing EPCM *)
IF (Other instructions accessing EPCM)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (DST) is pointing to a PT_REG or PT_TCS or PT_SS_FIRST or PT_SS_REST *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS)
  and (EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
    THEN #PF(DS:RCX); FI;

TMP_SECS := Get_SECS_ADDRESS();

IF (DS:RBX does not resolve to TMP_SECS)
    THEN #GP(0); FI;

(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUTES.INIT *)
IF ( (Other instruction accessing MRENCLAVE) or (Other instructions checking or updating the initialized state of the SECS))
```

```
    THEN #GP(0); FI;

(* Calculate enclave offset *)
TMP_ENCLAVEOFFSET := EPCM(DS:RCX).ENCLAVEADDRESS - TMP_SECS.BASEADDR;
TMP_ENCLAVEOFFSET := TMP_ENCLAVEOFFSET + (DS:RCX & 0FFFH)

(* Add EEXTEND message and offset to MRENCLAVE *)
TMPUPDATEFIELD[63:0] := 00444E4554584545H; // "EEXTEND"
TMPUPDATEFIELD[127:64] := TMP_ENCLAVEOFFSET;
TMPUPDATEFIELD[511:128] := 0; // 48 bytes
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, TMPUPDATEFIELD)
INC enclave's MRENCLAVE update counter;

(*Add 256 bytes to MRENCLAVE, 64 byte at a time *)
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[511:0] );
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[1023: 512] );
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[1535: 1024] );
TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[2047: 1536] );
INC enclave's MRENCLAVE update counter by 4;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If the address in RBX is outside the DS segment limit.
	If RBX points to an SECS page which is not the SECS of the data chunk.
	If the address in RCX is outside the DS segment limit.
	If RCX points to a memory location not 256Byte-aligned.
	If another instruction is accessing MRENCLAVE.
	If another instruction is checking or updating the SECS.
	If the enclave is already initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RBX points to a non-EPC page.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a non-EPC page.
	If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0)	If RBX is non-canonical form.
	If RBX points to an SECS page which is not the SECS of the data chunk.
	If RCX is non-canonical form.
	If RCX points to a memory location not 256 Byte-aligned.
	If another instruction is accessing MRENCLAVE.
	If another instruction is checking or updating the SECS.
	If the enclave is already initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RBX points to a non-EPC page.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a non-EPC page.
	If the address in RCX points to an invalid EPC page.

EINIT—Initialize an Enclave for Execution

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLS[EINIT]	IR	V/V	SGX1	This leaf function initializes the enclave and makes it ready to execute enclave code.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EINIT (In)	Error code (Out)	Address of SIGSTRUCT (In)	Address of SECS (In)	Address of EINITTOKEN (In)

Description

This leaf function is the final instruction executed in the enclave build process. After EINIT, the MRENCLAVE measurement is complete, and the enclave is ready to start user code execution using the EENTER instruction.

EINIT takes the effective address of a SIGSTRUCT and EINITTOKEN. The SIGSTRUCT describes the enclave including MRENCLAVE, ATTRIBUTES, ISVSVN, a 3072 bit RSA key, and a signature using the included key. SIGSTRUCT must be populated with two values, q1 and q2. These are calculated using the formulas shown below:

q1 = floor(Signature^2 / Modulus);

q2 = floor((Signature^3 - q1 * Signature * Modulus) / Modulus);

The EINITTOKEN contains the MRENCLAVE, MRSIGNER, and ATTRIBUTES. These values must match the corresponding values in the SECS. If the EINITTOKEN was created with a debug launch key, the enclave must be in debug mode as well.

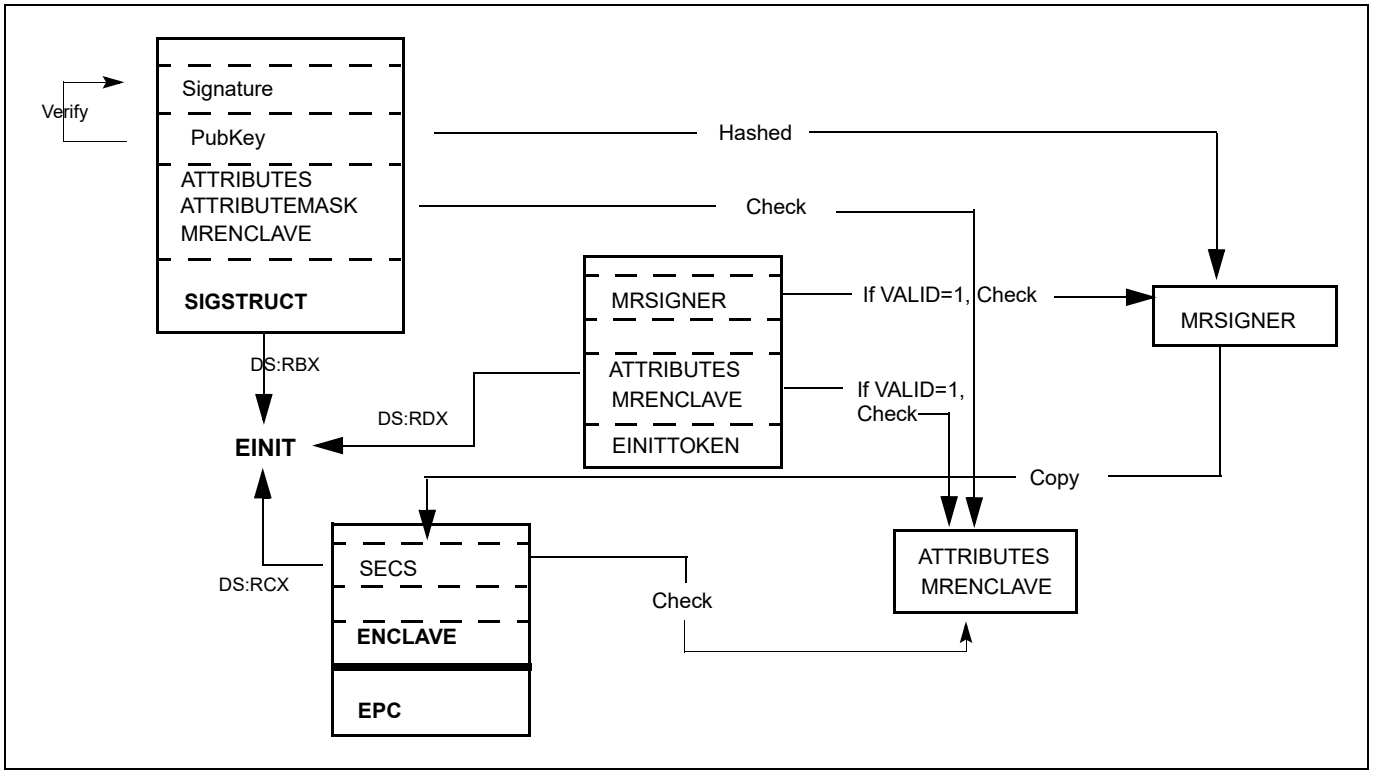


Figure 37-1. Relationships Between SECS, SIGSTRUCT and EINITTOKEN

EINIT Memory Parameter Semantics

SIGSTRUCT	SECS	EINITTOKEN
Access by non-Enclave	Read/Write access by Enclave	Access by non-Enclave

EINIT performs the following steps, which can be seen in Figure 37-1:

Validates that SIGSTRUCT is signed using the enclosed public key.

Checks that the completed computation of SECS.MRENCLAVE equals SIGSTRUCT.HASHENCLAVE.

Checks that no reserved bits are set to 1 in SIGSTRUCT.ATTRIBUTES and no reserved bits in SIGSTRUCT.ATTRIBUTESMASK are set to 0.

Checks that no controlled ATTRIBUTES bits are set in SIGSTRUCT.ATTRIBUTES unless the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.

Checks that SIGSTRUCT.ATTRIBUTES equals the result of logically and-ing SIGSTRUCT.ATTRIBUTEMASK with SECS.ATTRIBUTES.

If EINITTOKEN.VALID is 0, checks that the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.

If EINITTOKEN.VALID is 1, checks the validity of EINITTOKEN.

If EINITTOKEN.VALID is 1, checks that EINITTOKEN.MRENCLAVE equals SECS.MRENCLAVE.

If EINITTOKEN.VALID is 1 and EINITTOKEN.ATTRIBUTES.DEBUG is 1, SECS.ATTRIBUTES.DEBUG must be 1.

Commits SECS.MRENCLAVE, and sets SECS.MRSIGNER, SECS.ISVSVN, and SECS.ISVPRODID based on SIGSTRUCT.

Update the SECS as Initialized.

Periodically, EINIT polls for certain asynchronous events. If such an event is detected, it completes with failure code (ZF=1 and RAX = SGX_UNMASKED_EVENT), and RIP is incremented to point to the next instruction. These events includes external interrupts, non-maskable interrupts, system-management interrupts, machine checks, INIT signals, and the VMX-preemption timer. EINIT does not fail if the pending event is inhibited (e.g., external interrupts could be inhibited due to blocking by MOV SS blocking or by STI).

The following bits in RFLAGS are cleared: CF, PF, AF, OF, and SF. When the instruction completes with an error, RFLAGS.ZF is set to 1, and the corresponding error bit is set in RAX. If no error occurs, RFLAGS.ZF is cleared and RAX is set to 0.

The error codes are:

Table 37-25. EINIT Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	EINIT successful.
SGX_INVALID_SIG_STRUCT	If SIGSTRUCT contained an invalid value.
SGX_INVALID_ATTRIBUTE	If SIGSTRUCT contains an unauthorized attributes mask.
SGX_INVALID_MEASUREMENT	If SIGSTRUCT contains an incorrect measurement. If EINITTOKEN contains an incorrect measurement.
SGX_INVALID_SIGNATURE	If signature does not validate with enclosed public key.
SGX_INVALID_LICENSE	If license is invalid.
SGX_INVALID_CPUSVN	If license SVN is unsupported.
SGX_UNMASKED_EVENT	If an unmasked event is received before the instruction completes its operation.

Concurrency Restrictions

Table 37-26. Base Concurrency Restrictions of EINIT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EINIT	SECS [DS:RCX]	Shared	#GP	

Table 37-27. Additional Concurrency Restrictions of ENIT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EINIT	SECS [DS:RCX]	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EINIT Operational Flow

Name	Type	Size	Description
TMP_SIG	SIGSTRUCT	1808Bytes	Temp space for SIGSTRUCT.
TMP_TOKEN	EINITTOKEN	304Bytes	Temp space for EINITTOKEN.
TMP_MRENCLAVE		32Bytes	Temp space for calculating MRENCLAVE.
TMP_MRSIGNER		32Bytes	Temp space for calculating MRSIGNER.
CONTROLLED_ATTRIBUTES	ATTRIBUTES	16Bytes	Constant mask of all ATTRIBUTE bits that can only be set for authorized enclaves.
TMP_KEYDEPENDENCIES	Buffer	224Bytes	Temp space for key derivation.
TMP_EINITTOKENKEY		16Bytes	Temp space for the derived EINITTOKEN Key.
TMP_SIG_PADDING	PKCS Padding Buffer	352Bytes	The value of the top 352 bytes from the computation of Signature[3] modulo MRSIGNER.

```
(* make sure SIGSTRUCT and SECS are aligned *)
IF ( (DS:RBX is not 4KByte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* make sure the EINITTOKEN is aligned *)
IF (DS:RDX is not 512Byte Aligned)
    THEN #GP(0); FI;

(* make sure the SECS is inside the EPC *)
IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SIG[14463:0] := DS:RBX[14463:0]; // 1808 bytes
TMP_TOKEN[2423:0] := DS:RDX[2423:0]; // 304 bytes
```

```
(* Verify SIGSTRUCT Header. *)
IF ( (TMP_SIG.HEADER ≠ 06000000E10000000000010000000000h) or
     ((TMP_SIG.VENDOR ≠ 0) and (TMP_SIG.VENDOR ≠ 00008086h) ) or
     (TMP_SIG HEADER2 ≠ 01010000600000006000000001000000h) or
     (TMP_SIG.EXPONENT   ≠ 00000003h) or (Reserved space is not 0's) )
     THEN
          RFLAGS.ZF := 1;
          RAX := SGX_INVALID_SIG_STRUCT;
          GOTO EXIT;
FI;

(* Open "Event Window" Check for Interrupts. Verify signature using embedded public key, q1, and q2. Save upper 352 bytes of the
PKCS1.5 encoded message into the TMP_SIG_PADDING*)
IF (interrupt was pending) THEN
     RFLAGS.ZF := 1;
     RAX := SGX_UNMASKED_EVENT;
     GOTO EXIT;
FI
IF (signature failed to verify) THEN
     RFLAGS.ZF := 1;
     RAX := SGX_INVALID_SIGNATURE;
     GOTO EXIT;
FI;
(*Close "Event Window" *)

(* make sure no other Intel SGX instruction is modifying SECS*)
IF (Other instructions modifying SECS)
     THEN #GP(0); FI;

IF ( (EPCM(DS:RCX). VALID = 0) or (EPCM(DS:RCX).PT ≠ PT_SECS) )
     THEN #PF(DS:RCX); FI;

(* Verify ISVFAMILYID is not used on an enclave with KSS disabled *)
IF ((TMP_SIG.ISVFAMILYID != 0) AND (DS:RCX.ATTRIBUTES.KSS == 0))
     THEN
       RFLAGS.ZF := 1;
       RAX := SGX_INVALID_SIG_STRUCT;
       GOTO EXIT;
FI;

(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUTES.INIT *)
IF ( (Other instruction modifying MRENCLAVE) or (Other instructions modifying the SECS's Initialized state))
     THEN #GP(0); FI;

(* Calculate finalized version of MRENCLAVE *)
(* SHA256 algorithm requires one last update that compresses the length of the hashed message into the output SHA256 digest *)
TMP_ENCLAVE := SHA256FINAL( (DS:RCX).MRENCLAVE, enclave's MRENCLAVE update count *512);

(* Verify MRENCLAVE from SIGSTRUCT *)
IF (TMP_SIG.ENCLAVEHASH ≠ TMP_MRENCLAVE)
     RFLAGS.ZF := 1;
     RAX := SGX_INVALID_MEASUREMENT;
     GOTO EXIT;
FI;
```

```
TMP_MRSIGNER := SHA256(TMP_SIG.MODULUS)

(* if controlled ATTRIBUTES are set, SIGSTRUCT must be signed using an authorized key *)
CONTROLLED_ATTRIBUTES := 0000000000000020H;
IF ( ( (DS:RCX.ATTRIBUTES & CONTROLLED_ATTRIBUTES) ≠ 0) and (TMP_MRSIGNER ≠ IA32_SGXLEPUBKEYHASH) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_ATTRIBUTE;
    GOTO EXIT;
FI;

(* Verify SIGSTRUCT.ATTRIBUTE requirements are met *)
IF ( (DS:RCX.ATTRIBUTES & TMP_SIG.ATTRIBUTEMASK) ≠ (TMP_SIG.ATTRIBUTE & TMP_SIG.ATTRIBUTEMASK) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_ATTRIBUTE;
    GOTO EXIT;
FI;

( *Verify SIGSTRUCT.MISCSELECT requirements are met *)
IF ( (DS:RCX.MISCSELECT & TMP_SIG.MISCMASK) ≠ (TMP_SIG.MISCSELECT & TMP_SIG.MISCMASK) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ATTRIBUTE;
    GOTO EXIT
FI;

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    IF ( DS:RCX.CET_ATTRIBUTES & TMP_SIG.CET_ATTRIBUTES_MASK ≠ TMP_SIG.CET_ATTRIBUTES &
     TMP_SIG.CET_ATTRIB-UTES_MASK )
        THEN
            RFLAGS.ZF := 1;
            RAX := SGX_INVALID_ATTRIBUTE;
            GOTO EXIT
    FI;
FI;

(* If EINITTOKEN.VALID[0] is 0, verify the enclave is signed by an authorized key *)
IF (TMP_TOKEN.VALID[0] = 0)
    IF (TMP_MRSIGNER ≠ IA32_SGXLEPUBKEYHASH)
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_EINITTOKEN;
        GOTO EXIT;
    FI;
    GOTO COMMIT;
FI;

(* Debug Launch Enclave cannot launch Production Enclaves *)
IF ( (DS:RDX.MASKEDATTRIBUTESLE.DEBUG = 1) and (DS:RCX.ATTRIBUTES.DEBUG = 0) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINITTOKEN;
    GOTO EXIT;
FI;
```

```
(* Check reserve space in EINIT token includes reserved regions and upper bits in valid field *)
IF (TMP_TOKEN reserved space is not clear)
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINITTOKEN;
    GOTO EXIT;
FI;

(* EINIT token must not have been created by a configuration beyond the current CPU configuration *)
IF (TMP_TOKEN.CPUSVN must not be a configuration beyond CR_CPUSVN)
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_CPUSVN;
    GOTO EXIT;
FI;

(* Derive Launch key used to calculate EINITTOKEN.MAC *)
HARDCODED_PKCS1_5_PADDING[15:0] := 0100H;
HARDCODED_PKCS1_5_PADDING[2655:16] := SignExtend330Byte(-1); // 330 bytes of 0FFH
HARDCODED_PKCS1_5_PADDING[2815:2656] := 2004000501020403650148866009060D30313000H;

TMP_KEYDEPENDENCIES.KEYNAME := EINITTOKEN_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
TMP_KEYDEPENDENCIES.ISVPRODID := TMP_TOKEN.ISVPRODIDLE;
TMP_KEYDEPENDENCIES.ISVSVN := TMP_TOKEN.ISVSVNLE;
TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_TOKEN.MASKEDATTRIBUTESLE;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
TMP_KEYDEPENDENCIES.MRSIGNER := IA32_SGXLEPUBKEYHASH;
TMP_KEYDEPENDENCIES.KEYID := TMP_TOKEN.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN := TMP_TOKEN.CPUSVNLE;
TMP_KEYDEPENDENCIES.MISCSELECT := TMP_TOKEN.MASKEDMISCSELECTLE;
TMP_KEYDEPENDENCIES.MISCMASK := 0;
TMP_KEYDEPENDENCIES.PADDING := HARDCODED_PKCS1_5_PADDING;
TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
TMP_KEYDEPENDENCIES.CONFIGID := 0;
TMP_KEYDEPENDENCIES.CONFIGSVN := 0;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1))
    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_TOKEN.CET_MASKED_ATTRIBUTES_ LE;
    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
FI;

(* Calculate the derived key*)
TMP_EINITTOKENKEY := derivekey(TMP_KEYDEPENDENCIES);

(* Verify EINITTOKEN was generated using this CPU's Launch key and that it has not been modified since issuing by the Launch
Enclave. Only 192 bytes of EINITTOKEN are CMACed *)
IF (TMP_TOKEN.MAC ≠ CMAC(TMP_EINITTOKENKEY, TMP_TOKEN[1535:0] ) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINITTOKEN;
    GOTO EXIT;
FI;
```

```
(* Verify EINITTOKEN (RDX) is for this enclave *)
IF ( (TMP_TOKEN.MRENCLAVE ≠ TMP_MRENCLAVE) or (TMP_TOKEN.MRSIGNER ≠ TMP_MRSIGNER) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_MEASUREMENT;
    GOTO EXIT;
FI;

(* Verify ATTRIBUTES in EINITTOKEN are the same as the enclave's *)
IF (TMP_TOKEN.ATTRIBUTES ≠ DS:RCX.ATTRIBUTES)
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINIT_ATTRIBUTE;
    GOTO EXIT;
FI;

COMMIT:
(* Commit changes to the SECS; Set ISVPRODID, ISVSVN, MRSIGNER, INIT ATTRIBUTE fields in SECS (RCX) *)
DS:RCX.MRENCLAVE := TMP_MRENCLAVE;
(* MRSIGNER stores a SHA256 in little endian implemented natively on x86 *)
DS:RCX.MRSIGNER := TMP_MRSIGNER;
DS:RCX.ISVEXTPRODID := TMP_SIG.ISVEXTPRODID;
DS:RCX.ISVPRODID := TMP_SIG.ISVPRODID;
DS:RCX.ISVSVN := TMP_SIG.ISVSVN;
DS:RCX.ISVFAMILYID := TMP_SIG.ISVFAMILYID;
DS:RCX.PADDING := TMP_SIG_PADDING;

(* Mark the SECS as initialized *)
Update DS:RCX to initialized;

(* Set RAX and ZF for success*)
    RFLAGS.ZF := 0;
    RAX := 0;
EXIT:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

ZF is cleared if successful, otherwise ZF is set and RAX contains the error code. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand is not properly aligned.
	If another instruction is modifying the SECS.
	If the enclave is already initialized.
	If the SECS.MRENCLAVE is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If RCX does not resolve in an EPC page.
	If the memory address is not a valid, uninitialized SECS.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is not properly aligned.
	If another instruction is modifying the SECS.
	If the enclave is already initialized.
	If the SECS.MRENCLAVE is in use.

#PF(error code)	If a page fault occurs in accessing memory operands. If RCX does not resolve in an EPC page. If the memory address is not a valid, uninitialized SECS.

ELDB/ELDU/ELDBC/ELDUC—Load an EPC Page and Mark its State

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 07H ENCLS[ELDB]	IR	V/V	SGX1	This leaf function loads, verifies an EPC page and marks the page as blocked.
EAX = 08H ENCLS[ELDU]	IR	V/V	SGX1	This leaf function loads, verifies an EPC page and marks the page as unblocked.
EAX = 12H ENCLS[ELDBC]	IR	V/V	EAX[6]	This leaf function behaves lie ELDB but with improved conflict handling for oversubscription.
EAX = 13H ENCLS[ELDUC]	IR	V/V	EAX[6]	This leaf function behaves like ELDU but with improved conflict handling for oversubscription.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	ELDB/ELDU (In)	Return error code (Out)	Address of the PAGEINFO (In)	Address of the EPC page (In)	Address of the version-array slot (In)

Description

This leaf function copies a page from regular main memory to the EPC. As part of the copying process, the page is cryptographically authenticated and decrypted. This instruction can only be executed when current privilege level is 0.

The ELDB leaf function sets the BLOCK bit in the EPCM entry for the destination page in the EPC after copying. The ELDU leaf function clears the BLOCK bit in the EPCM entry for the destination page in the EPC after copying.

RBX contains the effective address of a PAGEINFO structure; RCX contains the effective address of the destination EPC page; RDX holds the effective address of the version array slot that holds the version of the page.

The ELDBC/ELDUC leafs are very similar to ELDB and ELDU. They provide an error code on the concurrency conflict for any of the pages which need to acquire a lock. These include the destination, SECS, and VA slot.

The table below provides additional information on the memory parameter of ELDB/ELDU leaf functions.

ELDB/ELDU/ELDBC/ELBUC Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.PCMD	PAGEINFO.SECS	EPCPAGE	Version-Array Slot
Non-enclave read access	Non-enclave read access	Non-enclave read access	Enclave read/write access	Read/Write access permitted by Enclave	Read/Write access permitted by Enclave

The error codes are:

Table 37-28. ELDB/ELDU/ELDBC/ELBUC Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	ELDB/ELDU successful.
SGX_MAC_COMPARE_FAIL	If the MAC check fails.

Concurrency Restrictions

Table 37-29. Base Concurrency Restrictions of ELDB/ELDU/ELDBC/ELBUC

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ELDB/ELDU	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA [DS:RDX]	Shared	#GP	
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP	
ELDBC/ELBUC	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	EPC_PAGE_CONFLICT_ERROR
	VA [DS:RDX]	Shared	SGX_EPC_PAGE_CONFLICT	
	SECS [DS:RBX]PAGEINFO.SECS	Shared	SGX_EPC_PAGE_CONFLICT	

Table 37-30. Additional Concurrency Restrictions of ELDB/ELDU/ELDBC/ELBUC

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ELDB/ELDU	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA [DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
ELDBC/ELBUC	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA [DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ELDB/ELDU/ELDBC/ELBUC Operational Flow

Name	Type	Size (Bits)	Description
TMP_SRCPGE	Memory page	4KBytes	
TMP_SECS	Memory page	4KBytes	
TMP_PCMD	PCMD	128 Bytes	
TMP_HEADER	MACHEADER	128 Bytes	
TMP_VER	UINT64	64	
TMP_MAC	UINT128	128	
TMP_PK	UINT128	128	Page encryption/MAC key.
SCRATCH_PCMD	PCMD	128 Bytes	

```
(* Check PAGEINFO and EPCPAGE alignment *)
IF ( (DS:RBX is not 32Byte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;
```

```
IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check VASLOT alignment *)
IF (DS:RDX is not 8Byte aligned)
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECS := DS:RBX.SECS;
TMP_PCMD := DS:RBX.PCMD;

(* Check alignment of PAGEINFO (RBX) linked parameters. Note: PCMD pointer is overlaid on top of PAGEINFO.SECINFO field *)
IF ( (DS:TMP_PCMD is not 128Byte aligned) or (DS:TMP_SRCPGE is not 4KByte aligned) )
    THEN #GP(0); FI;

(* Check concurrency of EPC by other Intel SGX instructions *)
IF (other instructions accessing EPC)
    THEN
    IF ((EAX==07h) OR (EAX==08h))   (* ELDB/ELDU *)
        THEN
            IF (<<VMX non-root operation>> AND
                    <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
                THEN
                    VMCS.Exit_reason := SGX_CONFLICT;
                    VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                    VMCS.Exit_qualification.error := 0;
                    VMCS.Guest-physical_address :=
                << translation of DS:RCX produced by paging >>;
                    VMCS.Guest-linear_address := DS:RCX;
                    Deliver VMEXIT;
                ELSE
                    #GP(0);
            FI;
        ELSE (* ELDBC/ELDUC *)
            IF (<<VMX non-root operation>> AND
                    <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
                THEN
                    VMCS.Exit_reason := SGX_CONFLICT;
                    VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_ERROR;
                    VMCS.Exit_qualification.error := SGX_EPC_PAGE_CONFLICT;
                    VMCS.Guest-physical_address :=
                << translation of DS:RCX produced by paging >>;
                    VMCS.Guest-linear_address := DS:RCX;
                    Deliver VMEXIT;
                ELSE
            RFLAGS.ZF := 1;
              RFLAGS.CF := 0;
            RAX := SGX_EPC_PAGE_CONFLICT;
            GOTO ERROR_EXIT;
            FI;
```

```
            FI;
FI;

(* Check concurrency of EPC and VASLOT by other Intel SGX instructions *)
IF (Other instructions modifying VA slot)
     THEN
          IF ((EAX==07h) OR (EAX==08h))   (* ELDB/ELDU *)
      #GP(0);
          FI;
   ELSE (* ELDBC/ELDUC *)
      RFLAGS.ZF := 1;
      RFLAGS.CF := 0;
      RAX := SGX_EPC_PAGE_CONFLICT;
      GOTO ERROR_EXIT;
FI;

(* Verify EPCM attributes of EPC page, VA, and SECS *)
IF (EPCM(DS:RCX).VALID = 1)
     THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RDX & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~0FFFH).PT ≠ PT_VA) )
     THEN #PF(DS:RDX); FI;

(* Copy PCMD into scratch buffer *)
SCRATCH_PCMD[1023: 0] := DS:TMP_PCMD[1023:0];

(* Zero out TMP_HEADER*)
TMP_HEADER[sizeof(TMP_HEADER)-1: 0] := 0;

TMP_HEADER.SECINFO := SCRATCH_PCMD.SECINFO;
TMP_HEADER.RSVD := SCRATCH_PCMD.RSVD;
TMP_HEADER.LINADDR := DS:RBX.LINADDR;

(* Verify various attributes of SECS parameter *)
IF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_REG) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_TCS) or
      (TMP_HEADER.SECINFO.FLAGS.PT = PT_TRIM) or
      (TMP_HEADER.SECINFO.FLAGS.PT = PT_SS_FIRST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1) or
      (TMP_HEADER.SECINFO.FLAGS.PT = PT_SS_REST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1))
     THEN
          IF ( DS:TMP_SECS is not 4KByte aligned)
               THEN #GP(0) FI;
          IF (DS:TMP_SECS does not resolve within an EPC)
               THEN #PF(DS:TMP_SECS) FI;
          IF ( Other instructions modifying SECS)
               THEN
                    IF ((EAX==07h) OR (EAX==08h))   (* ELDB/ELDU *)
                    #GP(0);
                    FI;
               ELSE (* ELDBC/ELDUC *)
                    RFLAGS.ZF := 1;
               RFLAGS.CF := 0;
               RAX := SGX_EPC_PAGE_CONFLICT;
               GOTO ERROR_EXIT;
          FI;
```

```
FI;

IF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_REG) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_TCS) or
      (TMP_HEADER.SECINFO.FLAGS.PT = PT_TRIM) or
      (TMP_HEADER.SECINFO.FLAGS.PT = PT_SS_FIRST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1) or
      (TMP_HEADER.SECINFO.FLAGS.PT = PT_SS_REST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1))
     THEN
          TMP_HEADER.EID := DS:TMP_SECS.EID;
     ELSE
          (* These pages do not have any parent, and hence no EID binding *)
          TMP_HEADER.EID := 0;
FI;

(* Copy 4KBytes SRCPGE to secure location *)
DS:RCX[32767: 0] := DS:TMP_SRCPGE[32767: 0];
TMP_VER := DS:RDX[63:0];

(* Decrypt and MAC page. AES_GCM_DEC has 2 outputs, {plain text, MAC} *)
(* Parameters for AES_GCM_DEC {Key, Counter, ..} *)
{DS:RCX, TMP_MAC} := AES_GCM_DEC(CR_BASE_PK, TMP_VER << 32, TMP_HEADER, 128, DS:RCX, 4096);

IF ( (TMP_MAC ≠ DS:TMP_PCMD.MAC) )
     THEN
          RFLAGS.ZF := 1;
          RAX := SGX_MAC_COMPARE_FAIL;
          GOTO ERROR_EXIT;
FI;

(* Check version before committing *)
IF (DS:RDX ≠ 0)
     THEN #GP(0);
     ELSE
          DS:RDX := TMP_VER;
FI;

(* Commit EPCM changes *)
EPCM(DS:RCX).PT := TMP_HEADER.SECINFO.FLAGS.PT;
EPCM(DS:RCX).RWX := TMP_HEADER.SECINFO.FLAGS.RWX;
EPCM(DS:RCX).PENDING := TMP_HEADER.SECINFO.FLAGS.PENDING;
EPCM(DS:RCX).MODIFIED := TMP_HEADER.SECINFO.FLAGS.MODIFIED;
EPCM(DS:RCX).PR := TMP_HEADER.SECINFO.FLAGS.PR;
EPCM(DS:RCX).ENCLAVEADDRESS := TMP_HEADER.LINADDR;

IF ( ((EAX = 07H) or (EAX = 12H)) and (TMP_HEADER.SECINFO.FLAGS.PT is NOT PT_SECS or PT_VA))
     THEN
          EPCM(DS:RCX).BLOCKED := 1;
     ELSE
          EPCM(DS:RCX).BLOCKED := 0;
FI;

IF (TMP_HEADER.SECINFO.FLAGS.PT is PT_SECS)
     << store translation of DS:RCX produced by paging in SECS(DS:RCX).ENCLAVECONTEXT >>
FI;
```

```
EPCM(DS:RCX). VALID := 1;

RAX := 0;
RFLAGS.ZF := 0;

ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if unsuccessful, otherwise cleared and RAX returns error code. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the instruction's EPC resource is in use by others.
	If the instruction fails to verify MAC.
	If the version-array slot is in use.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand expected to be in EPC does not resolve to an EPC page.
	If one of the EPC memory operands has incorrect page type.
	If the destination EPC page is already valid.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If the instruction's EPC resource is in use by others.
	If the instruction fails to verify MAC.
	If the version-array slot is in use.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand expected to be in EPC does not resolve to an EPC page.
	If one of the EPC memory operands has incorrect page type.
	If the destination EPC page is already valid.

EMODPR—Restrict the Permissions of an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0EH ENCLS[EMODPR]	IR	V/V	SGX2	This leaf function restricts the access rights associated with a EPC page in an initialized enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EMODPR (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function restricts the access rights associated with an EPC page in an initialized enclave. THE RWX bits of the SECINFO parameter are treated as a permissions mask; supplying a value that does not restrict the page permissions will have no effect. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODPR leaf function.

EMODPR Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave

The instruction faults if any of the following:

EMODPR Faulting Conditions

The operands are not properly aligned.	If unsupported security attributes are set.
The Enclave is not initialized.	SECS is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page in the running enclave.
The EPC page is not valid.	

The error codes are:

Table 37-31. EMODPR Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	EMODPR successful.
SGX_PAGE_NOT_MODIFIABLE	The EPC page cannot be modified because it is in the PENDING or MODIFIED state.
SGX_EPC_PAGE_CONFLICT	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODT, or EWB.

Concurrency Restrictions

Table 37-32. Base Concurrency Restrictions of EMODPR

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EMODPR	Target [DS:RCX]	Shared	#GP	

Table 37-33. Additional Concurrency Restrictions of EMODPR

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODPR	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	

Operation

Temp Variables in EMODPR Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operand belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or
    (SCRATCH_SECINFO.FLAGS.R is 0 and SCRATCH_SECINFO.FLAGS.W is not 0) )
    THEN #GP(0); FI;

(* Check concurrency with SGX1 or SGX2 instructions on the EPC page *)
IF (SGX1 or other SGX2 instructions accessing EPC page)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID is 0 )
    THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_EPC_PAGE_CONFLICT;
        GOTO DONE;
FI;

IF (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_PAGE_NOT_MODIFIABLE;
```

```
        GOTO DONE;
FI;

IF (EPCM(DS:RCX).PT is not PT_REG)
    THEN #PF(DS:RCX); FI;

TMP_SECS := GET_SECS_ADDRESS

IF (TMP_SECS.ATTRIBUTES.INIT = 0)
  THEN #GP(0); FI;

(* Set the PR bit to indicate that permission restriction is in progress *)
EPCM(DS:RCX).PR := 1;

(* Update EPCM permissions *)
EPCM(DS:RCX).R := EPCM(DS:RCX).R & SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := EPCM(DS:RCX).W & SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := EPCM(DS:RCX).X & SCRATCH_SECINFO.FLAGS.X;

RFLAGS.ZF := 0;
RAX := 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if page is not modifiable or if other SGX2 instructions are executing concurrently, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

EMODT—Change the Type of an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0FH ENCLS[EMODT]	IR	V/V	SGX2	This leaf function changes the type of an existing EPC page.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EMODT (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function modifies the type of an EPC page. The security attributes are configured to prevent access to the EPC page at its new type until a corresponding invocation of the EACCEPT leaf confirms the modification. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODT leaf function.

EMODT Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave

The instruction faults if any of the following:

EMODT Faulting Conditions

The operands are not properly aligned.	If unsupported security attributes are set.
The Enclave is not initialized.	SECS is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page in the running enclave.
The EPC page is not valid.	

The error codes are:

Table 37-34. EMODT Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	EMODT successful.
SGX_PAGE_NOT_MODIFIABLE	The EPC page cannot be modified because it is in the PENDING or MODIFIED state.
SGX_EPC_PAGE_CONFLICT	Page is being written by EADD, EAUG, ECREATE, ELDU/B, EMODPR, or EWB.

Concurrency Restrictions

Table 37-35. Base Concurrency Restrictions of EMODT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EMODT	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_ CONFLICT	EPC_PAGE_CONFLICT_ERROR

Table 37-36. Additional Concurrency Restrictions of EMODT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODT	Target [DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	

Operation

Temp Variables in EMODT Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operand belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or
    !(SCRATCH_SECINFO.FLAGS.PT is PT_TCS or SCRATCH_SECINFO.FLAGS.PT is PT_TRIM) )
    THEN #GP(0); FI;

(* Check concurrency with SGX1 instructions on the EPC page *)
IF (other SGX1 instructions accessing EPC page)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_EPC_PAGE_CONFLICT;
        GOTO DONE;
FI;

IF (EPCM(DS:RCX).VALID is 0)
    THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_EPC_PAGE_CONFLICT;
        GOTO DONE;
```

```
FI;

IF (!(EPCM(DS:RCX).PT is PT_REG or
    ((EPCM(DS:RCX).PT is PT_TCS or PT_SS_FIRST or PT_SS_REST) and SCRATCH_SECINFO.FLAGS.PT is PT_TRIM)))
        THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).PENDING is not 0 or (EPCM(DS:RCX).MODIFIED is not 0) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_PAGE_NOT_MODIFIABLE;
        GOTO DONE;
FI;

TMP_SECS := GET_SECS_ADDRESS

IF (TMP_SECS.ATTRIBUTES.INIT = 0)
    THEN #GP(0); FI;

(* Update EPCM fields *)
EPCM(DS:RCX).PR := 0;
EPCM(DS:RCX).MODIFIED := 1;
EPCM(DS:RCX).R := 0;
EPCM(DS:RCX).W := 0;
EPCM(DS:RCX).X := 0;
EPCM(DS:RCX).PT := SCRATCH_SECINFO.FLAGS.PT;

RFLAGS.ZF := 0;
RAX := 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if page is not modifiable or if other SGX2 instructions are executing concurrently, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

EPA—Add Version Array

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0AH ENCLS[EPA]	IR	V/V	SGX1	This leaf function adds a Version Array to the EPC.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EPA (In)	PT_VA (In, Constant)	Effective address of the EPC page (In)

Description

This leaf function creates an empty version array in the EPC page whose logical address is given by DS:RCX, and sets up EPCM attributes for that page. At the time of execution of this instruction, the register RBX must be set to PT_VA.

The table below provides additional information on the memory parameter of EPA leaf function.

EPA Memory Parameter Semantics

EPCPAGE
Write access permitted by Enclave

Concurrency Restrictions

Table 37-37. Base Concurrency Restrictions of EPA

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EPA	VA [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION

Table 37-38. Additional Concurrency Restrictions of EPA

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EPA	VA [DS:RCX]	Concurrent	L	Concurrent		Concurrent	

Operation

```
IF (RBX ≠ PT_VA or DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check concurrency with other Intel SGX instructions *)
IF (Other Intel SGX instructions accessing the page)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
```

```
                THEN
                    VMCS.Exit_reason := SGX_CONFLICT;
                    VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                    VMCS.Exit_qualification.error := 0;
                    VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
                    VMCS.Guest-linear_address := DS:RCX;
                Deliver VMEXIT;
                ELSE
                    #GP(0);
            FI;
FI;

(* Check EPC page must be empty *)
IF (EPCM(DS:RCX). VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

(* Clears EPC page *)
DS:RCX[32767:0] := 0;

EPCM(DS:RCX).PT := PT_VA;
EPCM(DS:RCX).ENCLAVEADDRESS := 0;
EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;
EPCM(DS:RCX).RWX := 0;
EPCM(DS:RCX).VALID := 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If another Intel SGX instruction is accessing the EPC page.
	If RBX is not set to PT_VA.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If the EPC page is valid.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If another Intel SGX instruction is accessing the EPC page.
	If RBX is not set to PT_VA.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If the EPC page is valid.

ERDINFO—Read Type and Status Information About an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 10H ENCLS[ERDINFO]	IR	V/V	EAX[6]	This leaf function returns type and status information about an EPC page.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	ERDINFO (In)	Return error code (Out)	Address of a RDINFO structure (In)	Address of the destination EPC page (In)

Description

This instruction reads type and status information about an EPC page and returns it in a RDINFO structure. The STATUS field of the structure describes the status of the page and determines the validity of the remaining fields. The FLAGS field returns the EPCM permissions of the page; the page type; and the BLOCKED, PENDING, MODIFIED, and PR status of the page. For enclave pages, the ENCLAVECONTEXT field of the structure returns the value of SECS.ENCLAVECONTEXT. For non-enclave pages (e.g., VA) ENCLAVECONTEXT returns 0.

For invalid or non-EPC pages, the instruction returns an information code indicating the page's status, in addition to populating the STATUS field.

ERDINFO returns an error code if the destination EPC page is being modified by a concurrent SGX instruction.

RBX contains the effective address of a RDINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of ERDINFO leaf function.

ERDINFO Memory Parameter Semantics

RDINFO	EPCPAGE
Read/Write access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

ERDINFO Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A memory operand is not properly aligned.
DS segment is unusable (32b mode).	A page fault occurs in accessing memory operands.
A memory address is in a non-canonical form (64b mode).	

The error codes are:

Table 37-39. ERDINFO Return Value in RAX

Error Code	Value	Description
No Error	0	ERDINFO successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.
SGX_PG_INVLD		Target page is not a valid EPC page.
SGX_PG_NONEPC		Page is not an EPC page.

Concurrency Restrictions

Table 37-40. Base Concurrency Restrictions of ERDINFO

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ERDINFO	Target [DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT	

Table 37-41. Additional Concurrency Restrictions of ERDINFO

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ERDINFO	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ERDINFO Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.
TMP_RDINFO	Linear Address	64	Address of the RDINFO structure.

```
(* check alignment of RDINFO structure (RBX) *)
IF (DS:RBX is not 32Byte Aligned) THEN
   #GP(0); FI;

(* check alignment of the EPCPAGE (RCX) *)
IF (DS:RCX is not 4KByte Aligned) THEN
   #GP(0); FI;

(* check that EPCPAGE (DS:RCX) is the address of an EPC page *)
IF (DS:RCX does not resolve within EPC) THEN
   RFLAGS.CF := 1;
   RFLAGS.ZF := 0;
   RAX := SGX_PG_NONEPC;
   goto DONE;
FI;

(* Check the EPC page for concurrency *)
IF (EPC page is being modified) THEN
   RFLAGS.ZF = 1;
   RFLAGS.CF = 0;
   RAX = SGX_EPC_PAGE_CONFLICT;
   goto DONE;
FI;

(* check page validity *)
IF (EPCM(DS:RCX).VALID = 0) THEN
   RFLAGS.CF = 1;
```

```
    RFLAGS.ZF = 0;
    RAX = SGX_PG_INVLD;
    goto DONE;
FI;

(* clear the fields of the RDINFO structure *)
TMP_RDINFO := DS:RBX;
TMP_RDINFO.STATUS := 0;
TMP_RDINFO.FLAGS := 0;
TMP_RDINFO.ENCLAVECONTEXT := 0;

(* store page info in RDINFO structure *)
TMP_RDINFO.FLAGS.RWX := EPCM(DS:RCX).RWX;
TMP_RDINFO.FLAGS.PENDING := EPCM(DS:RCX).PENDING;
TMP_RDINFO.FLAGS.MODIFIED := EPCM(DS:RCX).MODIFIED;
TMP_RDINFO.FLAGS.PR := EPCM(DS:RCX).PR;
TMP_RDINFO.FLAGS.PAGE_TYPE := EPCM(DS:RCX).PAGE_TYPE;
TMP_RDINFO.FLAGS.BLOCKED := EPCM(DS:RCX).BLOCKED;

(* read SECS.ENCLAVECONTEXT for enclave child pages *)
IF ((EPCM(DS:RCX).PAGE_TYPE = PT_REG) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_TCS) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_TRIM) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_SS_FIRST) or
    (EPCM(DS:RCX).PAGE_TYPE = PT_SS_REST)
   ) THEN
    TMP_SECS := Address of SECS for (DS:RCX);
    TMP_RDINFO.ENCLAVECONTEXT := SECS(TMP_SECS).ENCLAVECONTEXT;
FI;

(* populate enclave information for SECS pages *)
IF (EPCM(DS:RCX).PAGE_TYPE = PT_SECS) THEN
    IF ((VMX non-root mode) and
        (ENABLE_EPC_VIRTUALIZATION_EXTENSIONS Execution Control = 1)
       ) THEN
        TMP_RDINFO.STATUS.CHILDPRESENT :=
                        ((SECS(DS:RCX).CHLDCNT ≠ 0) or
                          SECS(DS:RCX).VIRTCHILDCNT ≠ 0);
    ELSE
        TMP_RDINFO.STATUS.CHILDPRESENT := (SECS(DS:RCX).CHLDCNT ≠ 0);
        TMP_RDINFO.STATUS.VIRTCHILDPRESENT :=
                        (SECS(DS:RCX).VIRTCHILDCNT ≠ 0);
        TMP_RDINFO.ENCLAVECONTEXT := SECS(DS_RCX).ENCLAVECONTEXT;
    FI;
FI;

RAX := 0;
RFLAGS.ZF := 0;
RFLAGS.CF := 0;

DONE:
(* clear flags *)
RFLAGS.PF := 0;
RFLAGS.AF := 0;
```

```
RFLAGS.OF := 0;
RFLAGS.SF := ? 0;
```

Flags Affected

ZF is set if ERDINFO fails due to concurrent operation with another SGX instruction; otherwise cleared.

CF is set if page is not a valid EPC page or not an EPC page; otherwise cleared.

PF, AF, OF and SF are cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If DS segment is unusable.
	If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands.

EREMOVE—Remove a page from the EPC

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 03H ENCLS[EREMOVE]	IR	V/V	SGX1	This leaf function removes a page from the EPC.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	EREMOVE (In)	Return error code (Out)	Effective address of the EPC page (In)

Description

This leaf function causes an EPC page to be un-associated with its SECS and be marked as unused. This instruction leaf can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create linear address. Segment override is not supported.

The instruction fails if the operand is not properly aligned or does not refer to an EPC page or the page is in use by another thread, or other threads are running in the enclave to which the page belongs. In addition the instruction fails if the operand refers to an SECS with associations.

EREMOVE Memory Parameter Semantics

EPCPAGE
Write access permitted by Enclave

The instruction faults if any of the following:

EREMOVE Faulting Conditions

The memory operand is not properly aligned.	The memory operand does not resolve in an EPC page.
Refers to an invalid SECS.	Refers to an EPC page that is locked by another thread.
Another Intel SGX instruction is accessing the EPC page.	RCX does not contain an effective address of an EPC page.
the EPC page refers to an SECS with associations.	

The error codes are:

Table 37-42. EREMOVE Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	EREMOVE successful.
SGX_CHILD_PRESENT	If the SECS still have enclave pages loaded into EPC.
SGX_ENCLAVE_ACT	If there are still logical processors executing inside the enclave.

Concurrency Restrictions

Table 37-43. Base Concurrency Restrictions of EREMOVE

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EREMOVE	Target [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION

Table 37-44. Additional Concurrency Restrictions of EREMOVE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EREMOVE	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EREMOVE Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.

```
IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve to an EPC page)
    THEN #PF(DS:RCX); FI;

TMP_SECS := Get_SECS_ADDRESS();

(* Check the EPC page for concurrency *)
IF (EPC page being referenced by another Intel SGX instruction)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
                VMCS.Guest-linear_address := DS:RCX;
            Deliver VMEXIT;
            ELSE
                #GP(0);
        FI;
FI;

(* if DS:RCX is already unused, nothing to do*)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PT = PT_TRIM AND EPCM(DS:RCX).MODIFIED = 0))
    THEN GOTO DONE;
FI;
```

```
IF ( (EPCM(DS:RCX).PT = PT_VA) OR
     ((EPCM(DS:RCX).PT = PT_TRIM) AND (EPCM(DS:RCX).MODIFIED = 0)) )
     THEN
          EPCM(DS:RCX).VALID := 0;
          GOTO DONE;
FI;

IF (EPCM(DS:RCX).PT = PT_SECS)
     THEN
          IF (DS:RCX has an EPC page associated with it)
               THEN
                    RFLAGS.ZF := 1;
                    RAX := SGX_CHILD_PRESENT;
                    GOTO ERROR_EXIT;
          FI;
          (* treat SECS as having a child page when VIRTCHILDCNT is non-zero *)
          IF (<<in VMX non-root operation>> AND
         <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>> AND
         (SECS(DS:RCX).VIRTCHILDCNT ≠ 0))
               THEN
                    RFLAGS.ZF := 1;
                    RAX := SGX_CHILD_PRESENT
                    GOTO ERROR_EXIT
          FI;
          EPCM(DS:RCX).VALID := 0;
          GOTO DONE;
FI;

IF (Other threads active using SECS)
     THEN
          RFLAGS.ZF := 1;
          RAX := SGX_ENCLAVE_ACT;
          GOTO ERROR_EXIT;
FI;

IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM) or
 (EPCM(DS:RCX).PT is PT_SS_FIRST) or (EPCM(DS:RCX).PT is PT_SS_REST))
     THEN
          EPCM(DS:RCX).VALID := 0;
          GOTO DONE;
FI;

DONE:
RAX := 0;
RFLAGS.ZF := 0;

ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if unsuccessful, otherwise cleared and RAX returns error code. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If another Intel SGX instruction is accessing the page.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If the memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If another Intel SGX instruction is accessing the page.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the memory operand is not an EPC page.

ETRACK—Activates EBLOCK Checks

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0CH ENCLS[ETRACK]	IR	V/V	SGX1	This leaf function activates EBLOCK checks.

Instruction Operand Encoding

Op/En	EAX		RCX
IR	ETRACK (In)	Return error code (Out)	Pointer to the SECS of the EPC page (In)

Description

This leaf function provides the mechanism for hardware to track that software has completed the required TLB address clears successfully. The instruction can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page.

The table below provides additional information on the memory parameter of ETRACK leaf function.

ETRACK Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 37-45. ETRACK Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	ETRACK successful.
SGX_PREV_TRK_INCMPL	All processors did not complete the previous shoot-down sequence.

Concurrency Restrictions

Table 37-46. Base Concurrency Restrictions of ETRACK

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ETRACK	SECS [DS:RCX]	Shared	#GP	

Table 37-47. Additional Concurrency Restrictions of ETRACK

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ETRACK	SECS [DS:RCX]	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE _CONFLICT

Operation

```
IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check concurrency with other Intel SGX instructions *)
IF (Other Intel SGX instructions using tracking facility on this SECS)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := TRACKING_RESOURCE_CONFLICT;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := SECS(TMP_SECS).ENCLAVECONTEXT;
                VMCS.Guest-linear_address := 0;
            Deliver VMEXIT;
            ELSE
                #GP(0);
        FI;
FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).PT ≠ PT_SECS)
    THEN #PF(DS:RCX); FI;

(* All processors must have completed the previous tracking cycle*)
IF ( (DS:RCX).TRACKING ≠ 0) )
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := TRACKING_REFERENCE_CONFLICT;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := SECS(TMP_SECS).ENCLAVECONTEXT;
                VMCS.Guest-linear_address := 0;
            Deliver VMEXIT;
        FI;
    RFLAGS.ZF := 1;
        RAX := SGX_PREV_TRK_INCMPL;
        GOTO DONE;
    ELSE
        RAX := 0;
        RFLAGS.ZF := 0;
FI;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if SECS is in use or invalid, otherwise cleared. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If another thread is concurrently using the tracking facility on this SECS.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If the specified EPC resource is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.

ETRACKC—Activates EBLOCK Checks

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 11H ENCLS[ETRACKC]	IR	V/V	EAX[6]	This leaf function activates EBLOCK checks.

Instruction Operand Encoding

Op/En	EAX		RCX	
IR	ETRACK (In)	Return error code (Out)	Address of the destination EPC page (In, EA)	Address of the SECS page (In, EA)

Description

The ETRACKC instruction is thread safe variant of ETRACK leaf and can be executed concurrently with other CPU threads operating on the same SECS.

This leaf function provides the mechanism for hardware to track that software has completed the required TLB address clears successfully. The instruction can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page.

The table below provides additional information on the memory parameter of ETRACK leaf function.

ETRACKC Memory Parameter Semantics

EPCPAGE
Read/Write access permitted by Enclave

The error codes are:

Table 37-48. ETRACKC Return Value in RAX

Error Code	Value	Description
No Error	0	ETRACKC successful.
SGX_EPC_PAGE_CONFLICT	7	Failure due to concurrent operation of another SGX instruction.
SGX_PG_INVLD	6	Target page is not a VALID EPC page.
SGX_PREV_TRK_INCMPL	17	All processors did not complete the previous tracking sequence.
SGX_TRACK_NOT_REQUIRED	27	Target page type does not require tracking.

Concurrency Restrictions

Table 37-49. Base Concurrency Restrictions of ETRACKC

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ETRACKC	Target [DS:RCX]	Shared	SGX_EPC_PAGE_ CONFLICT	
	SECS implicit	Concurrent		

Table 37-50. Additional Concurrency Restrictions of ETRACKC

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ETRACKC	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS implicit	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT

Operation

Temp Variables in ETRACKC Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.

```
(* check alignment of EPCPAGE (RCX) *)
IF (DS:RCX is not 4KByte Aligned) THEN
#GP(0); FI;

(* check that EPCPAGE (DS:RCX) is the address of an EPC page *)
IF (DS:RCX does not resolve within an EPC) THEN
#PF(DS:RCX, PFEC.SGX); FI;

(* Check the EPC page for concurrency *)
IF (EPC page is being modified) THEN
   RFLAGS.ZF := 1;
   RFLAGS.CF := 0;
   RAX := SGX_EPC_PAGE_CONFLICT;
   goto DONE_POST_LOCK_RELEASE;
FI;

(* check to make sure the page is valid *)
IF (EPCM(DS:RCX).VALID = 0) THEN
   RFLAGS.ZF := 1;
   RFLAGS.CF := 0;
   RAX := SGX_PG_INVLD;
   GOTO DONE;
FI;

(* find out the target SECS page *)
IF (EPCM(DS:RCX).PT is PT_REG or PT_TCS or PT_TRIM or PT_SS_FIRST or PT_SS_REST) THEN
   TMP_SECS := Obtain SECS through EPCM(DS:RCX).ENCLAVESECS;
ELSE IF (EPCM(DS:RCX).PT is PT_SECS) THEN
   TMP_SECS := Obtain SECS through (DS:RCX);
ELSE
   RFLAGS.ZF := 0;
   RFLAGS.CF := 1;
   RAX := SGX_TRACK_NOT_REQUIRED;
   GOTO DONE;
FI;
```

```
(* Check concurrency with other Intel SGX instructions *)
IF (Other Intel SGX instructions using tracking facility on this SECS) THEN
    IF ((VMX non-root mode) and
    (ENABLE_EPC_VIRTUALIZATION_EXTENSIONS Execution Control = 1)) THEN
        VMCS.Exit_reason := SGX_CONFLICT;
        VMCS.Exit_qualification.code := TRACKING_RESOURCE_CONFLICT;
        VMCS.Exit_qualification.error := 0;
        VMCS.Guest-physical_address :=
            SECS(TMP_SECS).ENCLAVECONTEXT;
        VMCS.Guest-linear_address := 0;
        Deliver VMEXIT;
    FI;

  RFLAGS.ZF := 1;
  RFLAGS.CF := 0;
  RAX := SGX_EPC_PAGE_CONFLICT;
  GOTO DONE;
FI;
(* All processors must have completed the previous tracking cycle*)
IF ( (TMP_SECS).TRACKING ≠ 0) )
THEN
    IF ((VMX non-root mode) and
    (ENABLE_EPC_VIRTUALIZATION_EXTENSIONS Execution Control = 1)) THEN
        VMCS.Exit_reason := SGX_CONFLICT;
        VMCS.Exit_qualification.code := TRACKING_REFERENCE_CONFLICT;
        VMCS.Exit_qualification.error := 0;
        VMCS.Guest-physical_address :=
            SECS(TMP_SECS).ENCLAVECONTEXT;
        VMCS.Guest-linear_address := 0;
        Deliver VMEXIT;
    FI;

  RFLAGS.ZF := 1;
  RFLAGS.CF := 0;
  RAX := SGX_PREV_TRK_INCMPL;
  GOTO DONE;
FI;

RFLAGS.ZF := 0;
RFLAGS.CF := 0;
RAX := 0;

DONE:
(* clear flags *)
RFLAGS.PF,AF,OF,SF := 0;
```

Flags Affected

ZF is set if ETRACKC fails due to concurrent operations with another SGX instructions or target page is an invalid EPC page or tracking is not completed on SECS page; otherwise cleared.

CF is set if target page is not of a type that requires tracking; otherwise cleared.

PF, AF, OF and SF are cleared.

Protected Mode Exceptions

#GP(0)	If the memory operand violates access-control policies of DS segment. If DS segment is unusable. If the memory operand is not properly aligned.
#PF(error code)	If the memory operand expected to be in EPC does not resolve to an EPC page. If a page fault occurs in access memory operand.

64-Bit Mode Exceptions

#GP(0)	If a memory address is in a non-canonical form. If a memory operand is not properly aligned.
#PF(error code)	If the memory operand expected to be in EPC does not resolve to an EPC page. If a page fault occurs in access memory operand.

EWB—Invalidate an EPC Page and Write out to Main Memory

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0BH ENCLS[EWB]	IR	V/V	SGX1	This leaf function invalidates an EPC page and writes it out to main memory.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EWB (In)	Error code (Out)	Address of an PAGEINFO (In)	Address of the EPC page (In)	Address of a VA slot (In)

Description

This leaf function copies a page from the EPC to regular main memory. As part of the copying process, the page is cryptographically protected. This instruction can only be executed when current privilege level is 0.

The table below provides additional information on the memory parameter of EPA leaf function.

EWB Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.PCMD	EPCPAGE	VASLOT
Non-EPC R/W access	Non-EPC R/W access	Non-EPC R/W access	EPC R/W access	EPC R/W access

The error codes are:

Table 37-51. EWB Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	EWB successful.
SGX_PAGE_NOT_BLOCKED	If page is not marked as blocked.
SGX_NOT_TRACKED	If EWB is racing with ETRACK instruction.
SGX_VA_SLOT_OCCUPIED	Version array slot contained valid entry.
SGX_CHILD_PRESENT	Child page present while attempting to page out enclave.

Concurrency Restrictions

Table 37-52. Base Concurrency Restrictions of EWB

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EWB	Source [DS:RCX]	Exclusive	#GP	EPC_PAGE_CONFLICT_EXCEPTION
	VA [DS:RDX]	Shared	#GP	

Table 37-53. Additional Concurrency Restrictions of EWB

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EWB	Source [DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA [DS:RDX]	Concurrent		Concurrent		Exclusive	

Operation

Temp Variables in EWB Operational Flow

Name	Type	Size (Bytes)	Description
TMP_SRCPGE	Memory page	4096	
TMP_PCMD	PCMD	128	
TMP_SECS	SECS	4096	
TMP_BPEPOCH	UINT64	8	
TMP_BPREFCOUNT	UINT64	8	
TMP_HEADER	MAC Header	128	
TMP_PCMD_ENCLAVEID	UINT64	8	
TMP_VER	UINT64	8	
TMP_PK	UINT128	16	

```
IF ( (DS:RBX is not 32Byte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF (DS:RDX is not 8Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

(* EPCPAGE and VASLOT should not resolve to the same EPC page*)
IF (DS:RCX and DS:RDX resolve to the same EPC page)
    THEN #GP(0); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
(* Note PAGEINFO.PCMD is overlaid on top of PAGEINFO.SECINFO *)
TMP_PCMD := DS:RBX.PCMD;

If (DS:RBX.LINADDR ≠ 0) OR (DS:RBX.SECS ≠ 0)
    THEN #GP(0); FI;

IF ( (DS:TMP_PCMD is not 128Byte Aligned) or (DS:TMP_SRCPGE is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* Check for concurrent Intel SGX instruction access to the page *)
IF (Other Intel SGX instruction is accessing page)
    THEN
        IF (<<VMX non-root operation>> AND <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>>)
            THEN
                VMCS.Exit_reason := SGX_CONFLICT;
                VMCS.Exit_qualification.code := EPC_PAGE_CONFLICT_EXCEPTION;
                VMCS.Exit_qualification.error := 0;
                VMCS.Guest-physical_address := << translation of DS:RCX produced by paging >>;
```

```
                VMCS.Guest-linear_address := DS:RCX;
            Deliver VMEXIT;
            ELSE
                #GP(0);
        FI;
FI;

(*Check if the VA Page is being removed or changed*)
IF (VA Page is being modified)
    THEN #GP(0); FI;

(* Verify that EPCPAGE and VASLOT page are valid EPC pages and DS:RDX is VA *)
IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RDX & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~FFFH).PT is not PT_VA) )
    THEN #PF(DS:RDX); FI;

(* Perform page-type-specific exception checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM ) or
 (EPCM(DS:RCX).PT is PT_SS_FIRST ) or (EPCM(DS:RCX).PT is PT_SS_REST))
    THEN
        TMP_SECS = Obtain SECS through EPCM(DS:RCX)
    (* Check that EBLOCK has occurred correctly *)
    IF (EBLOCK is not correct)
        THEN #GP(0); FI;
FI;

RFLAGS.ZF,CF,PF,AF,OF,SF := 0;
RAX := 0;

(* Zero out TMP_HEADER*)
TMP_HEADER[ sizeof(TMP_HEADER) - 1 : 0] := 0;

(* Perform page-type-specific checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM )or
 (EPCM(DS:RCX).PT is PT_SS_FIRST ) or (EPCM(DS:RCX).PT is PT_SS_REST))
    THEN
        (* check to see if the page is evictable *)
        IF (EPCM(DS:RCX).BLOCKED = 0)
            THEN
                RAX := SGX_PAGE NOT_BLOCKED;
                RFLAGS.ZF := 1;
                GOTO ERROR_EXIT;
        FI;
        (* Check if tracking done correctly *)
        IF (Tracking not correct)
            THEN
                RAX := SGX_NOT_TRACKED;
                RFLAGS.ZF := 1;
                GOTO ERROR_EXIT;
        FI;

        (* Obtain EID to establish cryptographic binding between the paged-out page and the enclave *)
```

```
        TMP_HEADER.EID := TMP_SECS.EID;

        (* Obtain EID as an enclave handle for software *)
        TMP_PCMD_ENCLAVEID := TMP_SECS.EID;
    ELSE IF (EPCM(DS:RCX).PT is PT_SECS)
        (*check that there are no child pages inside the enclave *)
        IF (DS:RCX has an EPC page associated with it)
            THEN
                RAX := SGX_CHILD_PRESENT;
                RFLAGS.ZF := 1;
                GOTO ERROR_EXIT;
        FI:
        (* treat SECS as having a child page when VIRTCHILDCNT is non-zero *)
        IF (<<in VMX non-root operation>> AND
     <<ENABLE_EPC_VIRTUALIZATION_EXTENSIONS>> AND
     (SECS(DS:RCX).VIRTCHILDCNT ≠ 0))
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_CHILD_PRESENT;
                GOTO ERROR_EXIT;
        FI;
        TMP_HEADER.EID := 0;
        (* Obtain EID as an enclave handle for software *)
        TMP_PCMD_ENCLAVEID := (DS:RCX).EID;
    ELSE IF (EPCM(DS:RCX).PT is PT_VA)
        TMP_HEADER.EID := 0; // Zero is not a special value
        (* No enclave handle for VA pages*)
        TMP_PCMD_ENCLAVEID := 0;
FI;

TMP_HEADER.LINADDR := EPCM(DS:RCX).ENCLAVEADDRESS;
TMP_HEADER.SECINFO.FLAGS.PT := EPCM(DS:RCX).PT;
TMP_HEADER.SECINFO.FLAGS.RWX := EPCM(DS:RCX).RWX;
TMP_HEADER.SECINFO.FLAGS.PENDING := EPCM(DS:RCX).PENDING;
TMP_HEADER.SECINFO.FLAGS.MODIFIED := EPCM(DS:RCX).MODIFIED;
TMP_HEADER.SECINFO.FLAGS.PR := EPCM(DS:RCX).PR;

(* Encrypt the page, DS:RCX could be encrypted in place. AES-GCM produces 2 values, {ciphertext, MAC}. *)
(* AES-GCM input parameters: key, GCM Counter, MAC_HDR, MAC_HDR_SIZE, SRC, SRC_SIZE)*)
{DS:TMP_SRCPGE, DS:TMP_PCMD.MAC} := AES_GCM_ENC(CR_BASE_PK), (TMP_VER << 32),
    TMP_HEADER, 128, DS:RCX, 4096);

(* Write the output *)
Zero out DS:TMP_PCMD.SECINFO
DS:TMP_PCMD.SECINFO.FLAGS.PT := EPCM(DS:RCX).PT;
DS:TMP_PCMD.SECINFO.FLAGS.RWX := EPCM(DS:RCX).RWX;
DS:TMP_PCMD.SECINFO.FLAGS.PENDING := EPCM(DS:RCX).PENDING;
DS:TMP_PCMD.SECINFO.FLAGS.MODIFIED := EPCM(DS:RCX).MODIFIED;
DS:TMP_PCMD.SECINFO.FLAGS.PR := EPCM(DS:RCX).PR;
DS:TMP_PCMD.RESERVED := 0;
DS:TMP_PCMD.ENCLAVEID := TMP_PCMD_ENCLAVEID;
DS:RBX.LINADDR := EPCM(DS:RCX).ENCLAVEADDRESS;

(*Check if version array slot was empty *)
```

```
IF ([DS.RDX])
    THEN
        RAX := SGX_VA_SLOT_OCCUPIED
        RFLAGS.CF := 1;
FI;

(* Write version to Version Array slot *)
[DS.RDX] := TMP_VER;

(* Free up EPCM Entry *)
EPCM.(DS:RCX).VALID := 0;
ERROR_EXIT:
```

Flags Affected

ZF is set if page is not blocked, not tracked, or a child is present. Otherwise cleared.

CF is set if VA slot is previously occupied, Otherwise cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the EPC page and VASLOT resolve to the same EPC page.
	If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.
	If the tracking resource is in use.
	If the EPC page or the version array page is invalid.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If one of the EPC memory operands has incorrect page type.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If the EPC page and VASLOT resolve to the same EPC page.
	If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.
	If the tracking resource is in use.
	If the EPC page or the version array page in invalid.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If one of the EPC memory operands has incorrect page type.

37.4 INTEL® SGX USER LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLU instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of the implicitly-encoded register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EACCEPT—Accept Changes to an EPC Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 05H ENCLU[EACCEPT]	IR	V/V	SGX2	This leaf function accepts changes made by system software to an EPC page in the running enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EACCEPT (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function accepts changes to a page in the running enclave by verifying that the security attributes specified in the SECINFO match the security attributes of the page in the EPCM. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EACCEPT leaf function.

EACCEPT Memory Parameter Semantics

SECINFO	EPCPAGE (Destination)
Read access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EACCEPT Faulting Conditions

The operands are not properly aligned.	RBX does not contain an effective address in an EPC page in the running enclave.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page in the running enclave.
The EPC page is not valid.	Page type is PT_REG and MODIFIED bit is 0.
SECINFO contains an invalid request.	Page type is PT_TCS or PT_TRIM and PENDING bit is 0 and MODIFIED bit is 1.
If security attributes of the SECINFO page make the page inaccessible.	

The error codes are:

Table 37-54. EACCEPT Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	EACCEPT successful.
SGX_PAGE_ATTRIBUTES_MISMATCH	The attributes of the target EPC page do not match the expected values.
SGX_NOT_TRACKED	The OS did not complete an ETRACK on the target page.

Concurrency Restrictions

Table 37-55. Base Concurrency Restrictions of EACCEPT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EACCEPT	Target [DS:RCX]	Shared	#GP	
	SECINFO [DS:RBX]	Concurrent		

Table 37-56. Additional Concurrency Restrictions of EACCEPT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EACCEPT	Target [DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EACCEPT Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Effective Address	32/64	Physical address of SECS to which EPC operands belongs.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX is not within CR_ELRANGE)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF ( (EPCM(DS:RBX &~FFFH).VALID = 0) or (EPCM(DS:RBX &~FFFH).R = 0) or (EPCM(DS:RBX &~FFFH).PENDING ≠ 0) or
    (EPCM(DS:RBX &~FFFH).MODIFIED ≠ 0) or (EPCM(DS:RBX &~FFFH).BLOCKED ≠ 0) or
    (EPCM(DS:RBX &~FFFH).PT ≠ PT_REG) or (EPCM(DS:RBX &~FFFH).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RBX &~FFFH).ENCLAVEADDRESS ≠ (DS:RBX & FFFH)) )
    THEN #PF(DS:RBX); FI;

(* Copy 64 bytes of contents *)
SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero )
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;
```

```
IF (DS:RCX is not within CR_ELRANGE)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check that the combination of requested PT, PENDING and MODIFIED is legal *)
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 0 )
    THEN
        IF (NOT (((SCRATCH_SECINFO.FLAGS.PT is PT_REG) and
         ((SCRATCH_SECINFO.FLAGS.PR is 1) or
         (SCRATCH_SECINFO.FLAGS.PENDING is 1)) and
         (SCRATCH_SECINFO.FLAGS.MODIFIED is 0)) or
         ((SCRATCH_SECINFO.FLAGS.PT is PT_TCS or PT_TRIM) and
         (SCRATCH_SECINFO.FLAGS.PR is 0) and
         (SCRATCH_SECINFO.FLAGS.PENDING is 0) and
         (SCRATCH_SECINFO.FLAGS.MODIFIED is 1) )))
            THEN #GP(0); FI
    ELSE
        IF (NOT (((SCRATCH_SECINFO.FLAGS.PT is PT_REG) AND
         ((SCRATCH_SECINFO.FLAGS.PR is 1) OR
         (SCRATCH_SECINFO.FLAGS.PENDING is 1)) AND
         (SCRATCH_SECINFO.FLAGS.MODIFIED is 0)) OR
         ((SCRATCH_SECINFO.FLAGS.PT is PT_TCS OR PT_TRIM) AND
         (SCRATCH_SECINFO.FLAGS.PENDING is 0) AND
         (SCRATCH_SECINFO.FLAGS.MODIFIED is 1) AND
         (SCRATCH_SECINFO.FLAGS.PR is 0)) OR
         ((SCRATCH_SECINFO.FLAGS.PT is PT_SS_FIRST or PT_SS_REST) AND
         (SCRATCH_SECINFO.FLAGS.PENDING is 1) AND
         (SCRATCH_SECINFO.FLAGS.MODIFIED is 0) AND
         (SCRATCH_SECINFO.FLAGS.PR is 0))))
            THEN #GP(0); FI;
    FI;

(* Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID is 0) or (EPCM(DS:RCX).BLOCKED is not 0) or
  ((EPCM(DS:RCX).PT is not PT_REG) and (EPCM(DS:RCX).PT is not PT_TCS) and (EPCM(DS:RCX).PT is not PT_TRIM)
  and (EPCM(DS:RCX).PT is not PT_SS_FIRST) and (EPCM(DS:RCX).PT is not PT_SS_REST)) or
  (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS))
    THEN #PF((DS:RCX); FI;

(* Check the destination EPC page for concurrency *)
IF ( EPC page in use )
    THEN #GP(0); FI;

(* Re-Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID is 0) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
    THEN #PF(DS:RCX); FI;

(* Verify that accept request matches current EPC page settings *)
IF ( (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX) or (EPCM(DS:RCX).PENDING ≠ SCRATCH_SECINFO.FLAGS.PENDING) or
    (EPCM(DS:RCX).MODIFIED ≠ SCRATCH_SECINFO.FLAGS.MODIFIED) or (EPCM(DS:RCX).R ≠ SCRATCH_SECINFO.FLAGS.R) or
    (EPCM(DS:RCX).W ≠ SCRATCH_SECINFO.FLAGS.W) or (EPCM(DS:RCX).X ≠ SCRATCH_SECINFO.FLAGS.X) or
    (EPCM(DS:RCX).PT ≠ SCRATCH_SECINFO.FLAGS.PT) )
```

```
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_PAGE_ATTRIBUTES_MISMATCH;
        GOTO DONE;
FI;
(* Check that all required threads have left enclave *)
IF (Tracking not correct)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_NOT_TRACKED;
        GOTO DONE;
FI;

(* Get pointer to the SECS to which the EPC page belongs *)
TMP_SECS = << Obtain physical address of SECS through EPCM(DS:RCX)>>
(* For TCS pages, perform additional checks *)
IF (SCRATCH_SECINFO.FLAGS.PT = PT_TCS)
    THEN
        IF (DS:RCX.RESERVED ≠ 0) #GP(0); FI;

    (* Check that TCS.FLAGS.DBGOPTIN, TCS stack, and TCS status are correctly initialized *)
    (* check that TCS.PREVSSP is 0 *)
    IF ( ((DS:RCX).FLAGS.DBGOPTIN is not 0) or ((DS:RCX).CSSA ≥ (DS:RCX).NSSA) or ((DS:RCX).AEP is not 0) or ((DS:RCX).STATE is not 0)
or ((CPUID.(EAX=07H, ECX=0H):ECX[CET_SS] = 1) AND ((DS:RCX).PREVSSP != 0)))
        THEN #GP(0); FI;

    (* Check consistency of FS & GS Limit *)
    IF ( (TMP_SECS.ATTRIBUTES.MODE64BIT is 0) and ((DS:RCX.FSLIMIT & FFFH ≠ FFFH) or (DS:RCX.GSLIMIT & FFFH ≠ FFFH)) )
        THEN #GP(0); FI;
FI;

(* Clear PENDING/MODIFIED flags to mark accept operation complete *)
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;

(* Clear EAX and ZF to indicate successful completion *)
RFLAGS.ZF := 0;
RAX := 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if page cannot be accepted, otherwise cleared. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If EPC page has incorrect page type or security attributes.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If EPC page has incorrect page type or security attributes.

EACCEPTCOPY—Initialize a Pending Page

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 07H ENCLU[EACCEPTCOPY]	IR	V/V	SGX2	This leaf function initializes a dynamically allocated EPC page from another page in the EPC.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EACCEPTCOPY (In)	Return Error Code (Out)	Address of a SECINFO (In)	Address of the destination EPC page (In)	Address of the source EPC page (In)

Description

This leaf function copies the contents of an existing EPC page into an uninitialized EPC page (created by EAUG). After initialization, the instruction may also modify the access rights associated with the destination EPC page. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX and RDX each contain the effective address of an EPC page. The table below provides additional information on the memory parameter of the EACCEPTCOPY leaf function.

EACCEPTCOPY Memory Parameter Semantics

SECINFO	EPCPAGE (Destination)	EPCPAGE (Source)
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EACCEPTCOPY Faulting Conditions

The operands are not properly aligned.	If security attributes of the SECINFO page make the page inaccessible.
The EPC page is locked by another thread.	If security attributes of the source EPC page make the page inaccessible.
The EPC page is not valid.	RBX does not contain an effective address in an EPC page in the running enclave.
SECINFO contains an invalid request.	RCX/RDX does not contain an effective address of an EPC page in the running enclave.

The error codes are:

Table 37-57. EACCEPTCOPY Return Value in RAX

Error Code (see Table 37-4)	Description
No Error	EACCEPTCOPY successful.
SGX_PAGE_ATTRIBUTES_MISMATCH	The attributes of the target EPC page do not match the expected values.

Concurrency Restrictions

Table 37-58. Base Concurrency Restrictions of EACCEPTCOPY

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EACCEPTCOPY	Target [DS:RCX]	Concurrent		
	Source [DS:RDX]	Concurrent		
	SECINFO [DS:RBX]	Concurrent		

Table 37-59. Additional Concurrency Restrictions of EACCEPTCOPY

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EACCEPTCOPY	Target [DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	Source [DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EACCEPTCOPY Operational Flow

Name	Type	Size (bits)	Description
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF ( (DS:RCX is not 4KByte Aligned) or (DS:RDX is not 4KByte Aligned) )
    THEN #GP(0); FI;

IF ((DS:RBX is not within CR_ELRANGE) or (DS:RCX is not within CR_ELRANGE) or (DS:RDX is not within CR_ELRANGE))
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

IF ( (EPCM(DS:RBX &~FFFH).VALID = 0) or (EPCM(DS:RBX &~FFFH).R = 0) or (EPCM(DS:RBX &~FFFH).PENDING ≠ 0) or
    (EPCM(DS:RBX &~FFFH).MODIFIED ≠ 0) or (EPCM(DS:RBX &~FFFH).BLOCKED ≠ 0) or (EPCM(DS:RBX &~FFFH).PT ≠ PT_REG) or
    (EPCM(DS:RBX &~FFFH).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RBX &~FFFH).ENCLAVEADDRESS ≠ DS:RBX) )
    THEN #PF(DS:RBX); FI;
```

```
(* Copy 64 bytes of contents *)
SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
IF ( (SCRATCH_SECINFO reserved fields are not zero ) or (SCRATCH_SECINFO.FLAGS.R=0) AND(SCRATCH_SECINFO.FLAGS.W≠0 ) or
     (SCRATCH_SECINFO.FLAGS.PT is not PT_REG) )
     THEN #GP(0); FI;

(* Check security attributes of the source EPC page *)
IF ( (EPCM(DS:RDX).VALID = 0) or (EPCM(DS:RCX).R = 0) or (EPCM(DS:RDX).PENDING ≠ 0) or (EPCM(DS:RDX).MODIFIED ≠ 0) or
     (EPCM(DS:RDX).BLOCKED ≠ 0) or (EPCM(DS:RDX).PT ≠ PT_REG) or (EPCM(DS:RDX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
     (EPCM(DS:RDX).ENCLAVEADDRESS ≠ DS:RDX))
     THEN #PF(DS:RDX); FI;

(* Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 1) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
     (EPCM(DS:RDX).BLOCKED ≠ 0) or (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
     THEN
          RFLAGS.ZF := 1;
          RAX := SGX_PAGE_ATTRIBUTES_MISMATCH;
          GOTO DONE;
FI;

(* Check the destination EPC page for concurrency *)
IF (destination EPC page in use )
     THEN #GP(0); FI;

(* Re-Check security attributes of the destination EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 1) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
     (EPCM(DS:RCX).R ≠ 1) or (EPCM(DS:RCX).W ≠ 1) or (EPCM(DS:RCX).X ≠ 0) or
     (EPCM(DS:RCX).PT ≠ SCRATCH_SECINFO.FLAGS.PT) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
     (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX))
     THEN
          RFLAGS.ZF := 1;
          RAX := SGX_PAGE_ATTRIBUTES_MISMATCH;
          GOTO DONE;
FI;

(* Copy 4KBbytes form the source to destination EPC page*)
DS:RCX[32767:0] := DS:RDX[32767:0];

(* Update EPCM permissions *)
EPCM(DS:RCX).R := SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PENDING := 0;

RFLAGS.ZF := 0;
RAX := 0;

DONE:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if page is not modifiable, otherwise cleared. Clears CF, PF, AF, OF, SF

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If EPC page has incorrect page type or security attributes.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If EPC page has incorrect page type or security attributes.

EENTER—Enters an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLU[EENTER]	IR	V/V	SGX1	This leaf function is used to enter an enclave.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	
IR	EENTER (In)	Content of RBX.CSSA (Out)	Address of a TCS (In)	Address of AEP (In)	Address of IP following EENTER (Out)

Description

The ENCLU[EENTER] instruction transfers execution to an enclave. At the end of the instruction, the logical processor is executing in enclave mode at the RIP computed as EnclaveBase + TCS.OENTRY. If the target address is not within the CS segment (32-bit) or is not canonical (64-bit), a #GP(0) results.

EENTER Memory Parameter Semantics

TCS
Enclave access

EENTER is a serializing instruction. The instruction faults if any of the following occurs:

Address in RBX is not properly aligned.	Any TCS.FLAGS's must-be-zero bit is not zero.
TCS pointed to by RBX is not valid or available or locked.	Current 32/64 mode does not match the enclave mode in SECS.ATTRIBUTES.MODE64.
The SECS is in use.	Either of TCS-specified FS and GS segment is not a subsets of the current DS segment.
Any one of DS, ES, CS, SS is not zero.	If XSAVE available, CR4.OSXSAVE = 0, but SECS.ATTRIBUTES.XFRM ≠ 3.
CR4.OSFXSR ≠ 1.	If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCR0.

The following operations are performed by EENTER:

- RSP and RBP are saved in the current SSA frame on EENTER and are automatically restored on EEXIT or interrupt.
- The AEP contained in RCX is stored into the TCS for use by AEXs.FS and GS (including hidden portions) are saved and new values are constructed using TCS.OFSBASE/GSBASE (32 and 64-bit mode) and TCS.OFSLIMIT/GSLIMIT (32-bit mode only). The resulting segments must be a subset of the DS segment.
- If CR4.OSXSAVE == 1, XCR0 is saved and replaced by SECS.ATTRIBUTES.XFRM. The effect of RFLAGS.TF depends on whether the enclave entry is opt-in or opt-out (see Section 39.1.2):
  - On opt-out entry, TF is saved and cleared (it is restored on EEXIT or AEX). Any attempt to set TF via a POPF instruction while inside the enclave clears TF (see Section 39.2.5).
  - On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 39.2.2).
- All code breakpoints that do not overlap with ELRANGE are also suppressed. If the entry is an opt-out entry, all code and data breakpoints that overlap with the ELRANGE are suppressed.
- On opt-out entry, a number of performance monitoring counters and behaviors are modified or suppressed (see Section 39.2.3):

— All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.

— PEBS is suppressed.

— AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set

— If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

Concurrency Restrictions

Table 37-60. Base Concurrency Restrictions of EENTER

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EENTER	TCS [DS:RBX]	Shared	#GP	

Table 37-61. Additional Concurrency Restrictions of EENTER

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EENTER	TCS [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EENTER Operational Flow

Name	Type	Size (Bits)	Description
TMP_FSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_GSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_FSLIMIT	Effective Address	32/64	Highest legal address in proposed FS segment.
TMP_GSLIMIT	Effective Address	32/64	Highest legal address in proposed GS segment.
TMP_XSIZE	integer	64	Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.
TMP_SSA_PAGE	Effective Address	32/64	Pointer used to iterate over the SSA pages in the current frame.
TMP_GPR	Effective Address	32/64	Address of the GPR area within the current SSA frame.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Make sure DS is usable, expand up *)
IF (TMP_MODE64 = 0 and (DS not usable or ( ( DS[S] = 1) and (DS[bit 11] = 0) and DS[bit 10] = 1) ) )
    THEN #GP(0); FI;

(* Check that CS, SS, DS, ES.base is 0 *)
IF (TMP_MODE64 = 0)
    THEN
        IF(CS.base ≠ 0 or DS.base ≠ 0) #GP(0); FI;
        IF(ES usable and ES.base ≠ 0) #GP(0); FI;
        IF(SS usable and SS.base ≠ 0) #GP(0); FI;
        IF(SS usable and SS.B = 0) #GP(0); FI;
```

```
FI;

IF (DS:RBX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

(* Check AEP is canonical*)
IF (TMP_MODE64 = 1 and (CS:RCX is not canonical) )
    THEN #GP(0); FI;

(* Check concurrency of TCS operation*)
IF (Other Intel SGX instructions is operating on TCS)
    THEN #GP(0); FI;

(* TCS verification *)
IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

IF ( (EPCM(DS:RBX).ENCLAVEADDRESS ≠ DS:RBX) or (EPCM(DS:RBX).PT ≠ PT_TCS) )
    THEN #PF(DS:RBX); FI;

IF ((EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1))
    THEN #PF(DS:RBX); FI;

IF ( (DS:RBX).OSSA is not 4KByte Aligned)
    THEN #GP(0); FI;

(* Check proposed FS and GS *)
IF ( ( (DS:RBX).OFSBASE is not 4KByte Aligned) or ( (DS:RBX).OGSBASE is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* Get the SECS for the enclave in which the TCS resides *)
TMP_SECS := Address of SECS for TCS;

(* Ensure that the FLAGS field in the TCS does not have any reserved bits set *)
IF ( ( (DS:RBX).FLAGS & FFFFFFFFFFFFFFFEH) ≠ 0)
    THEN #GP(0); FI;

(* SECS must exist and enclave must have previously been EINITted *)
IF (the enclave is not already initialized)
    THEN #GP(0); FI;

(* make sure the logical processor's operating mode matches the enclave *)
IF ( (TMP_MODE64 ≠ TMP_SECS.ATTRIBUTES.MODE64BIT) )
    THEN #GP(0); FI;

IF (CR4.OSFXSR = 0)
    THEN #GP(0); FI;
```

```
(* Check for legal values of SECS.ATTRIBUTES.XFRM *)
IF (CR4.OSXSAVE = 0)
    THEN
        IF (TMP_SECS.ATTRIBUTES.XFRM ≠ 03H) THEN #GP(0); FI;
    ELSE
        IF ( (TMP_SECS.ATTRIBUTES.XFRM & XCR0) ≠ TMP_SECS.ATTRIBUES.XFRM) THEN #GP(0); FI;
FI;

(* Make sure the SSA contains at least one more frame *)
IF ( (DS:RBX).CSSA ≥ (DS:RBX).NSSA)
    THEN #GP(0); FI;

(* Compute linear address of SSA frame *)
TMP_SSA := (DS:RBX).OSSA + TMP_SECS.BASEADDR + 4096 * TMP_SECS.SSAFRAMESIZE * (DS:RBX).CSSA;
TMP_XSIZE := compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);

FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
    (* Check page is read/write accessible *)
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort and deliver that fault;

    IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE).MODIFIED = 1))
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMP_SSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
        (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
        (EPCM(DS:TMP_SSA_PAGE).R = 0) or (EPCM(DS:TMP_SSA_PAGE).W = 0) )
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    CR_XSAVE_PAGE_n := Physical_Address(DS:TMP_SSA_PAGE);
ENDFOR

(* Compute address of GPR area*)
TMP_GPR := TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE - sizeof(GPRSGX_AREA);
If a fault occurs; release locks, abort and deliver that fault;

IF (DS:TMP_GPR does not resolve to EPC page)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).VALID = 0)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
    THEN #PF(DS:TMP_GPR); FI;
IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
    THEN #PF(DS:TMP_GPR); FI;
IF ( ( EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
    (EPCM(DS:TMP_GPR).ENCLAVESECS EPCM(DS:RBX).ENCLAVESECS) or
    (EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0) )
    THEN #PF(DS:TMP_GPR); FI;

IF (TMP_MODE64 = 0)
```

```
    THEN
        IF (TMP_GPR + (GPR_SIZE -1) is not in DS segment) THEN #GP(0); FI;
FI;

CR_GPR_PA := Physical_Address (DS: TMP_GPR);

(* Validate TCS.OENTRY *)
TMP_TARGET := (DS:RBX).OENTRY + TMP_SECS.BASEADDR;
IF (TMP_MODE64 = 1)
    THEN
        IF (TMP_TARGET is not canonical) THEN #GP(0); FI;
    ELSE
        IF (TMP_TARGET > CS limit) THEN #GP(0); FI;
FI;

(* Check proposed FS/GS segments fall within DS *)
IF (TMP_MODE64 = 0)
    THEN
        TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_FSLIMIT := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR + (DS:RBX).FSLIMIT;
        TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        TMP_GSLIMIT := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR + (DS:RBX).GSLIMIT;
        (* if FS wrap-around, make sure DS has no holes*)
        IF (TMP_FSLIMIT < TMP_FSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_FSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
        (* if GS wrap-around, make sure DS has no holes*)
        IF (TMP_GSLIMIT < TMP_GSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_GSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
    ELSE
        TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        IF ( (TMP_FSBASE is not canonical) or (TMP_GSBASE is not canonical))
            THEN #GP(0); FI;
FI;

(* Ensure the enclave is not already active and this thread is the only one using the TCS*)
IF (DS:RBX.STATE = ACTIVE)
    THEN #GP(0); FI;

TMP_IA32_U_CET := 0
TMP_SSP : = 0

IF CPUID.(EAX=12H, ECX=1):EAX[6] = 1
    THEN
        IF ( CR4.CET = 0 )
            THEN
```

```
                (* If part does not support CET or CET has not been enabled and enclave requires CET then fail *)
                IF ( TMP_SECS.CET_ATTRIBUTES ≠ 0 OR TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0 ) #GP(0); FI;
        FI;
        (* If indirect branch tracking or shadow stacks enabled but CET state save area is not 16B aligned then fail EENTER *)
        IF ( TMP_SECS.CET_ATTRIBUTES.SH_STK_EN = 1 OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN = 1 )
            THEN
                IF (DS:RBX.OCETSSA is not 16B aligned) #GP(0); FI;
        FI;

    IF (TMP_SECS.CET_ATTRIBUTES.SH_STK_EN OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN)
        THEN
            (* Setup CET state from SECS, note tracker goes to IDLE *)
            TMP_IA32_U_CET = TMP_SECS.CET_ATTRIBUTES;
            IF (TMP_IA32_U_CET.LEG_IW_EN = 1 AND TMP_IA32_U_CET.ENDBR_EN = 1 )
                THEN
                    TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.BASEADDR;
                    TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.CET_LEG_BITMAP_BASE;
            FI;

            (* Compute linear address of what will become new CET state save area and cache its PA *)
            TMP_CET_SAVE_AREA = DS:RBX.OCETSSA + TMP_SECS.BASEADDR + (DS:RBX.CSSA) * 16
            TMP_CET_SAVE_PAGE = TMP_CET_SAVE_AREA & ~0xFFF;

            Check the TMP_CET_SAVE_PAGE page is read/write accessible
            If fault occurs release locks, abort and deliver fault

            (* Read the EPCM VALID, PENDING, MODIFIED, BLOCKED and PT fields atomically *)
            IF ((DS:TMP_CET_SAVE_PAGE Does NOT RESOLVE TO EPC PAGE) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).VALID = 0) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).PENDING = 1) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).MODIFIED = 1) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).BLOCKED = 1) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).R = 0) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).W = 0) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVEADDRESS ≠ DS:TMP_CET_SAVE_PAGE) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).PT ≠ PT_SS_REST) OR
             (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS))
                THEN
                    #PF(DS:TMP_CET_SAVE_PAGE);
            FI;

            CR_CET_SAVE_AREA_PA := Physical address(DS:TMP_CET_SAVE_AREA)

            IF TMP_IA32_U_CET.SH_STK_EN = 1
                THEN
                    TMP_SSP = TCS.PREVSSP;
            FI;
    FI;
FI;

CR_ENCLAVE_MODE := 1;
CR_ACTIVE_SECS := TMP_SECS;
CR_ELRANGE := (TMPSECS.BASEADDR, TMP_SECS.SIZE);
```

```
(* Save state for possible AEXs *)
CR_TCS_PA := Physical_Address (DS:RBX);
CR_TCS_LA := RBX;
CR_TCS_LA.AEP := RCX;

(* Save the hidden portions of FS and GS *)
CR_SAVE_FS_selector := FS.selector;
CR_SAVE_FS_base := FS.base;
CR_SAVE_FS_limit := FS.limit;
CR_SAVE_FS_access_rights := FS.access_rights;
CR_SAVE_GS_selector := GS.selector;
CR_SAVE_GS_base := GS.base;
CR_SAVE_GS_limit := GS.limit;
CR_SAVE_GS_access_rights := GS.access_rights;

(* If XSAVE is enabled, save XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
IF (CR4.OSXSAVE = 1)
    CR_SAVE_XCR0 := XCR0;
    XCR0 := TMP_SECS.ATTRIBUTES.XFRM;
FI;

RCX := RIP;
RIP := TMP_TARGET;
RAX := (DS:RBX).CSSA;
(* Save the outside RSP and RBP so they can be restored on interrupt or EEXIT *)
DS:TMP_SSA.U_RSP := RSP;
DS:TMP_SSA.U_RBP := RBP;

(* Do the FS/GS swap *)
FS.base := TMP_FSBASE;
FS.limit := DS:RBX.FSLIMIT;
FS.type := 0001b;
FS.W := DS.W;
FS.S := 1;
FS.DPL := DS.DPL;
FS.G := 1;
FS.B := 1;
FS.P := 1;
FS.AVL := DS.AVL;
FS.L := DS.L;
FS.unusable := 0;
FS.selector := 0BH;

GS.base := TMP_GSBASE;
GS.limit := DS:RBX.GSLIMIT;
GS.type := 0001b;
GS.W := DS.W;
GS.S := 1;
GS.DPL := DS.DPL;
GS.G := 1;
GS.B := 1;
GS.P := 1;
GS.AVL := DS.AVL;
GS.L := DS.L;
```

```
GS.unusable := 0;
GS.selector := 0BH;

CR_DBGOPTIN := TCS.FLAGS.DBGOPTIN;
Suppress_all_code_breakpoints_that_are_outside_ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
        Suppress_all_code_breakpoints_that_overlap_with_ELRANGE;
        CR_SAVE_TF := RFLAGS.TF;
        RFLAGS.TF := 0;
        Suppress_monitor_trap_flag for the source of the execution of the enclave;
        Suppress any pending debug exceptions;
        Suppress any pending MTF VM exit;
    ELSE
        IF RFLAGS.TF = 1
            THEN pend a single-step #DB at the end of EENTER; FI;
        IF the “monitor trap flag” VM-execution control is set
            THEN pend an MTF VM exit at the end of EENTER; FI;
FI;

IF ((CPUID.(EAX=7H, ECX=0):EDX[CET_IBT] = 1) OR (CPUID.(EAX=7H, ECX=0):ECX[CET_SS] = 1)
    THEN
        (* Save enclosing application CET state into save registers *)
        CR_SAVE_IA32_U_CET := IA32_U_CET
        (* Setup enclave CET state *)
        IF CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1
            THEN
                CR_SAVE_SSP := SSP
                SSP := TMP_SSP
        FI;

        IA32_U_CET := TMP_IA32_U_CET;

FI;

Flush_linear_context;
Allow_front_end_to_begin_fetch_at_new_RIP;
```

Flags Affected

RFLAGS.TF is cleared on opt-out entry.

Protected Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If the enclave is not initialized.
	If part or all of the FS or GS segment specified by TCS is outside the DS segment or not properly aligned.
	If the thread is not in the INACTIVE state.
	If CS, DS, ES or SS bases are not all zero.
	If executed in enclave mode.
	If any reserved field in the TCS FLAG is set.
	If the target address is not within the CS segment.

	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
#PF(error code)	If a page fault occurs in accessing memory.
	If DS:RBX does not point to a valid TCS.
	If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If the enclave is not initialized.
	If the thread is not in the INACTIVE state.
	If CS, DS, ES or SS bases are not all zero.
	If executed in enclave mode.
	If part or all of the FS or GS segment specified by TCS is outside the DS segment or not properly aligned.
	If the target address is not canonical.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If DS:RBX does not point to a valid TCS.
	If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

EEXIT—Exits an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 04H ENCLU[EEXIT]	IR	V/V	SGX1	This leaf function is used to exit an enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EEXIT (In)	Target address outside the enclave (In)	Address of the current AEP (Out)

Description

The ENCLU[EEXIT] instruction exits the currently executing enclave and branches to the location specified in RBX. RCX receives the current AEP. If RBX is not within the CS (32-bit mode) or is not canonical (64-bit mode) a #GP(0) results.

EEXIT Memory Parameter Semantics

Target Address
Non-Enclave read and execute access

If RBX specifies an address that is inside the enclave, the instruction will complete normally. The fetch of the next instruction will occur in non-enclave mode, but will attempt to fetch from inside the enclave. This fetch returns a fixed data pattern.

If secrets are contained in any registers, it is responsibility of enclave software to clear those registers.

If XCR0 was modified on enclave entry, it is restored to the value it had at the time of the most recent EENTER or ERESUME.

If the enclave is opt-out, RFLAGS.TF is loaded from the value previously saved on EENTER.

Code and data breakpoints are unsuppressed.

Performance monitoring counters are unsuppressed.

Concurrency Restrictions

Table 37-62. Base Concurrency Restrictions of EEXIT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EEXIT		Concurrent		

Table 37-63. Additional Concurrency Restrictions of EEXIT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EEXIT		Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EEXIT Operational Flow

Name	Type	Size (Bits)	Description
TMP_RIP	Effective Address	32/64	Saved copy of CRIP for use when creating LBR.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF (TMP_MODE64 = 1)
    THEN
        IF (RBX is not canonical) THEN #GP(0); FI;
    ELSE
        IF (RBX > CS limit) THEN #GP(0); FI;
FI;

TMP_RIP := CRIP;
RIP := RBX;

(* Return current AEP in RCX *)
RCX := CR_TCS_PA.AEP;

(* Do the FS/GS swap *)
FS.selector := CR_SAVE_FS.selector;
FS.base := CR_SAVE_FS.base;
FS.limit := CR_SAVE_FS.limit;
FS.access_rights := CR_SAVE_FS.access_rights;
GS.selector := CR_SAVE_GS.selector;
GS.base := CR_SAVE_GS.base;
GS.limit := CR_SAVE_GS.limit;
GS.access_rights := CR_SAVE_GS.access_rights;

(* Restore XCR0 if needed *)
IF (CR4.OSXSAVE = 1)
    XCR0 := CR_SAVE__XCR0;
FI;

Unsuppress_all_code_breakpoints_that_are_outside_ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
        UnSuppress_all_code_breakpoints_that_overlap_with_ELRANGE;
        Restore suppressed breakpoint matches;
        RFLAGS.TF := CR_SAVE_TF;
        UnSuppress_montior_trap_flag;
        UnSuppress_LBR_Generation;
        UnSuppress_performance monitoring_activity;
        Restore performance monitoring counter AnyThread demotion to MyThread in enclave back to AnyThread
FI;

IF RFLAGS.TF = 1
    THEN Pend Single-Step #DB at the end of EEXIT;
FI;
```

```
IF the "monitor trap flag" VM-execution control is set
    THEN pend a MTF VM exit at the end of EEXIT;
FI;

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        (* Record PREVSSP *)
        IF (IA32_U_CET.SH_STK_EN == 1)
            THEN CR_TCS_PA.PREVSSP = SSP; FI;
FI;

IF ((CPUID.(EAX=7H, ECX=0):EDX[CET_IBT] = 1) OR (CPUID.(EAX=7, ECX=0):ECX[CET_SS] = 1)
    THEN
        (* Restore enclosing app's CET state from the save registers *)
        IA32_U_CET := CR_SAVE_IA32_U_CET;
        IF CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1
            THEN SSP := CR_SAVE_SSP; FI;

        (* Update enclosing app's TRACKER if enclosing app has indirect branch tracking enabled *)
        IF (CR4.CET = 1 AND IA32_U_CET.ENDBR_EN = 1)
            THEN
                IA32_U_CET.TRACKER := WAIT_FOR_ENDBRANCH;
                IA32_U_CET.SUPPRESS := 0
        FI;
FI;

CR_ENCLAVE_MODE := 0;
CR_TCS_PA.STATE := INACTIVE;

(* Assure consistent translations *)
Flush_linear_context;
```

Flags Affected

RFLAGS.TF is restored from the value previously saved in EENTER or ERESUME.

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If RBX is outside the CS segment.
#PF(error code)	If a page fault occurs in accessing memory.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If RBX is not canonical.
#PF(error code)	If a page fault occurs in accessing memory operands.

EGETKEY—Retrieves a Cryptographic Key

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLU[EGETKEY]	IR	V/V	SGX1	This leaf function retrieves a cryptographic key.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EGETKEY (In)	Return error code (Out)	Address to a KEYREQUEST (In)	Address of the OUTPUTDATA (In)

Description

The ENCLU[EGETKEY] instruction returns a 128-bit secret key from the processor specific key hierarchy. The register RBX contains the effective address of a KEYREQUEST structure, which the instruction interprets to determine the key being requested. The Requesting Keys section below provides a description of the keys that can be requested. The RCX register contains the effective address where the key will be returned. Both the addresses in RBX & RCX should be locations inside the enclave.

EGETKEY derives keys using a processor unique value to create a specific key based on a number of possible inputs. This instruction leaf can only be executed inside an enclave.

EEGETKEY Memory Parameter Semantics

KEYREQUEST	OUTPUTDATA
Enclave read access	Enclave write access

After validating the operands, the instruction determines which key is to be produced and performs the following actions:

- The instruction assembles the derivation data for the key based on the Table 37-64.
- Computes derived key using the derivation data and package specific value.
- Outputs the calculated key to the address in RCX.

The instruction fails with #GP(0) if the operands are not properly aligned. Successful completion of the instruction will clear RFLAGS.{ZF, CF, AF, OF, SF, PF}. The instruction returns an error code if the user tries to request a key based on an invalid CPUSVN or ISVSVN (when the user request is accepted, see the table below), requests a key for which it has not been granted the attribute to request, or requests a key that is not supported by the hardware. These checks may be performed in any order. Thus, an indication by error number of one cause (for example, invalid attribute) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same Enclave. The correctness of software should not rely on the order resulting from the checks documented in this section. In such cases the ZF flag is set and the corresponding error bit (SGX_INVALID_SVN, SGX_INVALID_ATTRIBUTE, SGX_INVALID_KEYNAME) is set in RAX and the data at the address specified by RCX is unmodified.

Requesting Keys

The KEYREQUEST structure (see Section 34.18.1) identifies the key to be provided. The Keyrequest.KeyName field identifies which type of key is requested.

Deriving Keys

Key derivation is based on a combination of the enclave specific values (see Table 37-64) and a processor key. Depending on the key being requested a field may either be included by definition or the value may be included from the KeyRequest. A "yes" in Table 37-64 indicates the value for the field is included from its default location, identified in the source row, and a "request" indicates the values for the field is included from its corresponding KeyRequest field.

Table 37-64. Key Derivation

	Key Name	Attributes	Owner Epoch	CPU SVN	ISV SVN	ISV PRODID	ISVEXT PRODID	ISVFAM ILYID	MRENCLAVE	MRSIGNER	CONFIG ID	CONFIGS VN	RAND
Source	Key Dependent Constant	Y := SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES; R := AttribMask & SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES;	CR_SGX OWNER EPOCH	Y := CPUSVN Register; R := Req.CPU SVN;	R := Req.ISV SVN;	SECS. ISVID	SECS.IS VEXTPR ODID	SECS.IS VFAMIL YID	SECS. MRENCLAVE	SECS. MRSIGNER	SECS.CO NFIGID	SECS.CO NFIGSVN	Req. KEYID
EINITTOKEN	Yes	Request	Yes	Request	Request	Yes	No	No	No	Yes	No	No	Request
Report	Yes	Yes	Yes	Yes	No	No	No	No	Yes	No	Yes	Yes	Request
Seal	Yes	Request	Yes	Request	Request	Request	Request	Request	Request	Request	Request	Request	Request
Provisioning	Yes	Request	No	Request	Request	Yes	No	No	No	Yes	No	No	Yes
Provisioning Seal	Yes	Request	No	Request	Request	Request	Request	Request	No	Yes	Request	Request	Yes

Keys that permit the specification of a CPU or ISV's code's, or enclave configuration's SVNs have additional requirements. The caller may not request a key for an SVN beyond the current CPU, ISV or enclave configuration's SVN, respectively.

Several keys are access controlled. Access to the Provisioning Key and Provisioning Seal key requires the enclave's ATTRIBUTES.PROVISIONKEY be set. The EINITTOKEN Key requires ATTRIBUTES.EINITTOKEN_KEY be set and SECS.MRSIGNER equal IA32_SGXLEPUBKEYHASH.

Some keys are derived based on a hardcode PKCS padding constant (352 byte string):

HARDCODED_PKCS1_5_PADDING[15:0] := 0100H;

HARDCODED_PKCS1_5_PADDING[2655:16] := SignExtend330Byte(-1); // 330 bytes of 0FFH

HARDCODED_PKCS1_5_PADDING[2815:2656] := 2004000501020403650148866009060D30313000H;

The error codes are:

Table 37-65. EGETKEY Return Value in RAX

Error Code (see Table 37-4)	Value	Description
No Error	0	EGETKEY successful.
SGX_INVALID_ATTRIBUTE		The KEYREQUEST contains a KEYNAME for which the enclave is not authorized.
SGX_INVALID_CPUSVN		If KEYREQUEST.CPUSVN is an unsupported platforms CPUSVN value.
SGX_INVALID_ISVSVN		If KEYREQUEST software SVN (ISVSVN or CONFIGSVN) is greater than the enclave's corresponding SVN.
SGX_INVALID_KEYNAME		If KEYREQUEST.KEYNAME is an unsupported value.

Concurrency Restrictions

Table 37-66. Base Concurrency Restrictions of EGETKEY

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EGETKEY	KEYREQUEST [DS:RBX]	Concurrent		
	OUTPUTDATA [DS:RCX]	Concurrent		

Table 37-67. Additional Concurrency Restrictions of EGETKEY

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EGETKEY	KEYREQUEST [DS:RBX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EGETKEY Operational Flow

Name	Type	Size (Bits)	Description
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.
TMP_KEYDEPENDENCIES			Temp space for key derivation.
TMP_ATTRIBUTES		128	Temp Space for the calculation of the sealable Attributes.
TMP_ISVEXTPRODID		16 bytes	Temp Space for ISVEXTPRODID.
TMP_ISVPRODID		2 bytes	Temp Space for ISVPRODID.
TMP_ISVFAMILYID		16 bytes	Temp Space for ISVFAMILYID.
TMP_CONFIGID		64 bytes	Temp Space for CONFIGID.
TMP_CONFIGSVN		2 bytes	Temp Space for CONFIGSVN.
TMP_OUTPUTKEY		128	Temp Space for the calculation of the key.

```
(* Make sure KEYREQUEST is properly aligned and inside the current enclave *)
IF ( (DS:RBX is not 512Byte aligned) or (DS:RBX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

(* Make sure DS:RBX is an EPC address and the EPC page is valid *)
IF ( (DS:RBX does not resolve to an EPC address) or (EPCM(DS:RBX).VALID = 0) )
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or
    (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH) ) or (EPCM(DS:RBX).R = 0) )
    THEN #PF(DS:RBX);
FI;

(* Make sure OUTPUTDATA is properly aligned and inside the current enclave *)
IF ( (DS:RCX is not 16Byte aligned) or (DS:RCX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

(* Make sure DS:RCX is an EPC address and the EPC page is valid *)
IF ( (DS:RCX does not resolve to an EPC address) or (EPCM(DS:RCX).VALID = 0) )
```

```
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).BLOCKED = 1)
    THEN #PF(DS:RCX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).W = 0) )
    THEN #PF(DS:RCX);
FI;

(* Verify RESERVED spaces in KEYREQUEST are valid *)
IF ( (DS:RBX).RESERVED ≠ 0) or (DS:RBX.KEYPOLICY.RESERVED ≠ 0) )
    THEN #GP(0); FI;

TMP_CURRENTSECS := CR_ACTIVE_SECS;

(* Verify that CONFIGSVN & New Policy bits are not used if KSS is not enabled *)
IF ((TMP_CURRENTSECS.ATTRIBUTES.KSS == 0) AND ((DS:RBX.KEYPOLICY & 0x003C ≠ 0) OR (DS:RBX.CONFIGSVN > 0)))
    THEN #GP(0); FI;
(* Determine which enclave attributes that must be included in the key. Attributes that must always be include INIT & DEBUG *)
REQUIRED_SEALING_MASK[127:0] := 00000000 00000000 00000000 00000003H;
TMP_ATTRIBUTES := (DS:RBX.ATTRIBUTEMASK | REQUIRED_SEALING_MASK) & TMP_CURRENTSECS.ATTRIBUTES;

(* Compute MISCSELECT fields to be included *)
TMP_MISCSELECT := DS:RBX.MISCMASK & TMP_CURRENTSECS.MISCSELECT

(* Compute CET_ATTRIBUTES fields to be included *)
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN TMP_CET_ATTRIBUTES := DS:RBX.CET_ATTRIBUTES_ MASK & TMP_CURRENTSECS.CET_ATTRIBUTES; FI;
TMP_KEYDEPENDENCIES := 0;

CASE (DS:RBX.KEYNAME)
    SEAL_KEY:
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CONFIGSVN > TMP_CURRENTSECS.CONFIGSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;

        (*Include enclave identity?*)
```

```
            TMP_MRENCLAVE := 0;
            IF (DS:RBX.KEYPOLICY.MRENCLAVE = 1)
                  THEN TMP_MRENCLAVE := TMP_CURRENTSECS.MRENCLAVE;
            FI;
            (*Include enclave author?*)
            TMP_MRSIGNER := 0;
            IF (DS:RBX.KEYPOLICY.MRSIGNER = 1)
                  THEN TMP_MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
            FI;
(* Include enclave product family ID? *)
   TMP_ISVFAMILYID := 0;
   IF (DS:RBX.KEYPOLICY.ISVFAMILYID = 1)
      THEN TMP_ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
            FI;

   (* Include enclave product ID? *)
   TMP_ISVPRODID := 0;
   IF (DS:RBX.KEYPOLICY.NOISVPRODID = 0)
      TMP_ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
            FI;

   (* Include enclave Config ID? *)
   TMP_CONFIGID := 0;
   TMP_CONFIGSVN := 0;
   IF (DS:RBX.KEYPOLICY.CONFIGID = 1)
      TMP_CONFIGID := TMP_CURRENTSECS.CONFIGID;
      TMP_CONFIGSVN := DS:RBX.CONFIGSVN;
            FI;

   (* Include enclave extended product ID? *)
   TMP_ISVEXTPRODID := 0;
   IF (DS:RBX.KEYPOLICY.ISVEXTPRODID = 1 )
      TMP_ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
   FI;

            //Determine values key is based on
            TMP_KEYDEPENDENCIES.KEYNAME := SEAL_KEY;
            TMP_KEYDEPENDENCIES.ISVFAMILYID := TMP_ISVFAMILYID;
            TMP_KEYDEPENDENCIES.ISVEXTPRODID := TMP_ISVEXTPRODID;
            TMP_KEYDEPENDENCIES.ISVPRODID := TMP_ISVPRODID;
            TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
            TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
            TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
            TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTEMASK;
            TMP_KEYDEPENDENCIES.MRENCLAVE := TMP_MRENCLAVE;
            TMP_KEYDEPENDENCIES.MRSIGNER := TMP_MRSIGNER;
            TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
            TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
            TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
            TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
            TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
            TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
            TMP_KEYDEPENDENCIES.KEYPOLICY := DS:RBX.KEYPOLICY;
            TMP_KEYDEPENDENCIES.CONFIGID := TMP_CONFIGID;
```

```
        TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CONFIGSVN;
        IF CPUID.(EAX=12H, ECX=1):EAX[6] = 1
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := DS:RBX.CET_ATTRIBUTES _MASK;
        FI;
        BREAK;
    REPORT_KEY:
        //Determine values key is based on
        TMP_KEYDEPENDENCIES.KEYNAME := REPORT_KEY;
        TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
        TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVSVN := 0;
        TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
        TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_CURRENTSECS.ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
        TMP_KEYDEPENDENCIES.MRENCLAVE := TMP_CURRENTSECS.MRENCLAVE;
        TMP_KEYDEPENDENCIES.MRSIGNER := 0;
        TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN := CR_CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING := HARDCODED_PKCS1_5_PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT := TMP_CURRENTSECS.MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK := 0;
        TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
        TMP_KEYDEPENDENCIES.CONFIGID := TMP_CURRENTSECS.CONFIGID;
        TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CURRENTSECS.CONFIGSVN;
        IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CURRENTSECS.CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
        FI;
        BREAK;
    EINITTOKEN_KEY:
        (* Check ENCLAVE has EINITTOKEN Key capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.EINITTOKEN_KEY = 0)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ATTRIBUTE;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
```

```
        (* Determine values key is based on *)
        TMP_KEYDEPENDENCIES.KEYNAME := EINITTOKEN_KEY;
        TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
        TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVPRODID := TMP_CURRENTSECS.ISVPRODID
        TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
        TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
        TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
        TMP_KEYDEPENDENCIES.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
        TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK := 0;
        TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
        TMP_KEYDEPENDENCIES.CONFIGID := 0;
        TMP_KEYDEPENDENCIES.CONFIGSVN := 0;
        IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
        FI;
        BREAK;
PROVISION_KEY:
(* Check ENCLAVE has PROVISIONING capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ATTRIBUTE;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
        (* Determine values key is based on *)
        TMP_KEYDEPENDENCIES.KEYNAME := PROVISION_KEY;
        TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
        TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
        TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
        TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := 0;
        TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
```

```
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTEMASK;
        TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
        TMP_KEYDEPENDENCIES.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID := 0;
        TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := 0;
        TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
        TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
        TMP_KEYDEPENDENCIES.CONFIGID := 0;
        IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
        FI;
        BREAK;
    PROVISION_SEAL_KEY:
        (* Check ENCLAVE has PROVISIONING capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ATTRIBUTE;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
(* Include enclave product family ID? *)
    TMP_ISVFAMILYID := 0;
    IF (DS:RBX.KEYPOLICY.ISVFAMILYID = 1)
        THEN TMP_ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
            FI;

    (* Include enclave product ID? *)
    TMP_ISVPRODID := 0;
    IF (DS:RBX.KEYPOLICY.NOISVPRODID = 0)
        TMP_ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
            FI;

    (* Include enclave Config ID? *)
    TMP_CONFIGID := 0;
    TMP_CONFIGSVN := 0;
    IF (DS:RBX.KEYPOLICY.CONFIGID = 1)
        TMP_CONFIGID := TMP_CURRENTSECS.CONFIGID;
```

```
        TMP_CONFIGSVN := DS:RBX.CONFIGSVN;
            FI;

    (* Include enclave extended product ID? *)
    TMP_ISVEXTPRODID := 0;
    IF (DS:RBX.KEYPOLICY.ISVEXTPRODID = 1)
        TMP_ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
    FI;

            (* Determine values key is based on *)
            TMP_KEYDEPENDENCIES.KEYNAME := PROVISION_SEAL_KEY;
            TMP_KEYDEPENDENCIES.ISVFAMILYID := TMP_ISVFAMILYID;
            TMP_KEYDEPENDENCIES.ISVEXTPRODID := TMP_ISVEXTPRODID;
            TMP_KEYDEPENDENCIES.ISVPRODID := TMP_ISVPRODID;
            TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
            TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := 0;
            TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
            TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTEMASK;
            TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
            TMP_KEYDEPENDENCIES.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
            TMP_KEYDEPENDENCIES.KEYID := 0;
            TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
            TMP_KEYDEPENDENCIES.CPUSVN := DS:RBX.CPUSVN;
            TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
            TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
            TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
            TMP_KEYDEPENDENCIES.KEYPOLICY := DS:RBX.KEYPOLICY;
            TMP_KEYDEPENDENCIES.CONFIGID := TMP_CONFIGID;
            TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CONFIGSVN;
            IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
                THEN
                    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
            FI;
            BREAK;
    DEFAULT:
            (* The value of KEYNAME is invalid *)
            RFLAGS.ZF := 1;
            RAX := SGX_INVALID_KEYNAME;
            GOTO EXIT:
ESAC;

(* Calculate the final derived key and output to the address in RCX *)
TMP_OUTPUTKEY := derivekey(TMP_KEYDEPENDENCIES);
DS:RCX[15:0] := TMP_OUTPUTKEY;
RAX := 0;
RFLAGS.ZF := 0;

EXIT:
RFLAGS.CF := 0;
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;
```

Flags Affected

ZF is cleared if successful, otherwise ZF is set. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the current enclave.
	If an effective address is not properly aligned.
	If an effective address is outside the DS segment limit.
	If KEYREQUEST format is invalid.
#PF(error code)	If a page fault occurs in accessing memory.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the current enclave.
	If an effective address is not properly aligned.
	If an effective address is not canonical.
	If KEYREQUEST format is invalid.
#PF(error code)	If a page fault occurs in accessing memory operands.

EMODPE—Extend an EPC Page Permissions

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 06H ENCLU[EMODPE]	IR	V/V	SGX2	This leaf function extends the access rights of an existing EPC page.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EMODPE (In)	Address of a SECINFO (In)	Address of the destination EPC page (In)

Description

This leaf function extends the access rights associated with an existing EPC page in the running enclave. THE RWX bits of the SECINFO parameter are treated as a permissions mask; supplying a value that does not extend the page permissions will have no effect. This instruction leaf can only be executed when inside the enclave.

RBX contains the effective address of a SECINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of the EMODPE leaf function.

EMODPE Memory Parameter Semantics

SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EMODPE Faulting Conditions

The operands are not properly aligned.	If security attributes of the SECINFO page make the page inaccessible.
The EPC page is locked by another thread.	RBX does not contain an effective address in an EPC page in the running enclave.
The EPC page is not valid.	RCX does not contain an effective address of an EPC page in the running enclave.
SECINFO contains an invalid request.	

Concurrency Restrictions

Table 37-68. Base Concurrency Restrictions of EMODPE

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EMODPE	Target [DS:RCX]	Concurrent		
	SECINFO [DS:RBX]	Concurrent		

Table 37-69. Additional Concurrency Restrictions of EMODPE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EMODPE	Target [DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EMODPE Operational Flow

Name	Type	Size (bits)	Description
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:RBX.

```
IF (DS:RBX is not 64Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF ((DS:RBX is not within CR_ELRANGE) or (DS:RCX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RBX).VALID = 0) or (EPCM(DS:RBX).R = 0) or (EPCM(DS:RBX).PENDING ≠ 0) or (EPCM(DS:RBX).MODIFIED ≠ 0) or
    (EPCM(DS:RBX).BLOCKED ≠ 0) or (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0xFFF)) )
    THEN #PF(DS:RBX); FI;

SCRATCH_SECINFO := DS:RBX;

(* Check for misconfigured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero )
    THEN #GP(0); FI;

(* Check security attributes of the EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 0) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
    (EPCM(DS:RCX).BLOCKED ≠ 0) or (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) )
    THEN #PF(DS:RCX); FI;

(* Check the EPC page for concurrency *)
IF (EPC page in use by another SGX2 instruction)
    THEN #GP(0); FI;

(* Re-Check security attributes of the EPC page *)
IF ( (EPCM(DS:RCX).VALID = 0) or (EPCM(DS:RCX).PENDING ≠ 0) or (EPCM(DS:RCX).MODIFIED ≠ 0) or
    (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or
    (EPCM(DS:RCX).ENCLAVEADDRESS ≠ DS:RCX))
    THEN #PF(DS:RCX); FI;

(* Check for misconfigured SECINFO flags*)
IF ( (EPCM(DS:RCX).R = 0) and (SCRATCH_SECINFO.FLAGS.R = 0) and (SCRATCH_SECINFO.FLAGS.W ≠ 0) )
    THEN #GP(0); FI;
```

```
(* Update EPCM permissions *)
EPCM(DS:RCX).R := EPCM(DS:RCX).R | SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := EPCM(DS:RCX).W | SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := EPCM(DS:RCX).X | SCRATCH_SECINFO.FLAGS.X;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is locked.
#PF(error code)	If a page fault occurs in accessing memory operands.

EREPORT—Create a Cryptographic Report of the Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLU[EREPORT]	IR	V/V	SGX1	This leaf function creates a cryptographic report of the enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX	RDX
IR	EREPORT (In)	Address of TARGETINFO (In)	Address of REPORTDATA (In)	Address where the REPORT is written to in an OUTPUTDATA (In)

Description

This leaf function creates a cryptographic REPORT that describes the contents of the enclave. This instruction leaf can only be executed when inside the enclave. The cryptographic report can be used by other enclaves to determine that the enclave is running on the same platform.

RBX contains the effective address of the MRENCLAVE value of the enclave that will authenticate the REPORT output, using the REPORT key delivered by EGETKEY command for that enclave. RCX contains the effective address of a 64-byte REPORTDATA structure, which allows the caller of the instruction to associate data with the enclave from which the instruction is called. RDX contains the address where the REPORT will be output by the instruction.

EREPORT Memory Parameter Semantics

TARGETINFO	REPORTDATA	OUTPUTDATA
Read access by Enclave	Read access by Enclave	Read/Write access by Enclave

This instruction leaf perform the following:

1. Validate the 3 operands (RBX, RCX, RDX) are inside the enclave.
2. Compute a report key for the target enclave, as indicated by the value located in RBX(TARGETINFO).
3. Assemble the enclave SECS data to complete the REPORT structure (including the data provided using the RCX (REPORTDATA) operand).
4. Computes a cryptographic hash over REPORT structure.
5. Add the computed hash to the REPORT structure.
6. Output the completed REPORT structure to the address in RDX (OUTPUTDATA).

The instruction fails if the operands are not properly aligned.

CR_REPORT_KEYID, used to provide key wearout protection, is populated with a statistically unique value on boot of the platform by a trusted entity within the SGX TCB.

The instruction faults if any of the following:

EREPORT Faulting Conditions

An effective address not properly aligned.	An memory address does not resolve in an EPC page.
If accessing an invalid EPC page.	If the EPC page is blocked.
May page fault.	

Concurrency Restrictions

Table 37-70. Base Concurrency Restrictions of EREPORT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EREPORT	TARGETINFO [DS:RBX]	Concurrent		
	REPORTDATA [DS:RCX]	Concurrent		
	OUTPUTDATA [DS:RDX]	Concurrent		

Table 37-71. Additional Concurrency Restrictions of EREPORT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EREPORT	TARGETINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	
	REPORTDATA [DS:RCX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA [DS:RDX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EREPORT Operational Flow

Name	Type	Size (bits)	Description
TMP_ATTRIBUTES		32	Physical address of SECS of the enclave to which source operand belongs.
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.
TMP_KEYDEPENDENCIES			Temp space for key derivation.
TMP_REPORTKEY		128	REPORTKEY generated by the instruction.
TMP_REPORT		3712	

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Address verification for TARGETINFO (RBX) *)
IF ( (DS:RBX is not 512Byte Aligned) or (DS:RBX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or
    (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH) ) or (EPCM(DS:RBX).R = 0) )
```

```
    THEN #PF(DS:RBX);
FI;

(* Verify RESERVED spaces in TARGETINFO are valid *)
IF (DS:RBX.RESERVED != 0)
    THEN #GP(0); FI;

(* Address verification for REPORTDATA (RCX) *)
IF ( (DS:RCX is not 128Byte Aligned) or (DS:RCX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).BLOCKED = 1)
    THEN #PF(DS:RCX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).R = 0) )
    THEN #PF(DS:RCX);
FI;

(* Address verification for OUTPUTDATA (RDX) *)
IF ( (DS:RDX is not 512Byte Aligned) or (DS:RDX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

IF (EPCM(DS:RDX).VALID = 0)
    THEN #PF(DS:RDX); FI;

IF (EPCM(DS:RDX).BLOCKED = 1)
    THEN #PF(DS:RDX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RDX).PT ≠ PT_REG) or (EPCM(DS:RDX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RDX).ENCLAVEADDRESS ≠ (DS:RDX & ~0FFFH) ) or (EPCM(DS:RDX).W = 0) )
    THEN #PF(DS:RDX);
FI;

(* REPORT MAC needs to be computed over data which cannot be modified *)
TMP_REPORT.CPUSVN := CR_CPUSVN;
TMP_REPORT.ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
TMP_REPORT.ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
TMP_REPORT.ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
TMP_REPORT.ISVSVN := TMP_CURRENTSECS.ISVSVN;
TMP_REPORT.ATTRIBUTES := TMP_CURRENTSECS.ATTRIBUTES;
TMP_REPORT.REPORTDATA := DS:RCX[511:0];
TMP_REPORT.MRENCLAVE := TMP_CURRENTSECS.MRENCLAVE;
```

```
TMP_REPORT.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
TMP_REPORT.MRRESERVED := 0;
TMP_REPORT.KEYID[255:0] := CR_REPORT_KEYID;
TMP_REPORT.MISCSELECT := TMP_CURRENTSECS.MISCSELECT;
TMP_REPORT.CONFIGID := TMP_CURRENTSECS.CONFIGID;
TMP_REPORT.CONFIGSVN := TMP_CURRENTSECS.CONFIGSVN;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN TMP_REPORT.CET_ATTRIBUTES := TMP_CURRENTSECS.CET_ATTRIBUTES; FI;

(* Derive the report key *)
TMP_KEYDEPENDENCIES.KEYNAME := REPORT_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
TMP_KEYDEPENDENCIES.ISVPRODID := 0;
TMP_KEYDEPENDENCIES.ISVSVN := 0;
TMP_KEYDEPENDENCIES.SGXOWNEREPOCH := CR_SGXOWNEREPOCH;
TMP_KEYDEPENDENCIES.ATTRIBUTES := DS:RBX.ATTRIBUTES;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
TMP_KEYDEPENDENCIES.MRENCLAVE := DS:RBX.MEASUREMENT;
TMP_KEYDEPENDENCIES.MRSIGNER := 0;
TMP_KEYDEPENDENCIES.KEYID := TMP_REPORT.KEYID;
TMP_KEYDEPENDENCIES.SEAL_KEY_FUSES := CR_SEAL_FUSES;
TMP_KEYDEPENDENCIES.CPUSVN := CR_CPUSVN;
TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
TMP_KEYDEPENDENCIES.MISCSELECT := DS:RBX.MISCSELECT;
TMP_KEYDEPENDENCIES.MISCMASK := 0;
TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
TMP_KEYDEPENDENCIES.CONFIGID := DS:RBX.CONFIGID;
TMP_KEYDEPENDENCIES.CONFIGSVN := DS:RBX.CONFIGSVN;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := DS:RBX.CET_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
FI;

(* Calculate the derived key*)
TMP_REPORTKEY := derivekey(TMP_KEYDEPENDENCIES);

(* call cryptographic CMAC function, CMAC data are not including MAC&KEYID *)
TMP_REPORT.MAC := cmac(TMP_REPORTKEY, TMP_REPORT[3071:0] );
DS:RDX[3455: 0] := TMP_REPORT;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If the address in RCS is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is not in the current enclave.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If RCX is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is not in the current enclave.
#PF(error code)	If a page fault occurs in accessing memory operands.

ERESUME—Re-Enters an Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 03H ENCLU[ERESUME]	IR	V/V	SGX1	This leaf function is used to re-enter an enclave after an interrupt.

Instruction Operand Encoding

Op/En	RAX	RBX	RCX
IR	ERESUME (In)	Address of a TCS (In)	Address of AEP (In)

Description

The ENCLU[ERESUME] instruction resumes execution of an enclave that was interrupted due to an exception or interrupt, using the machine state previously stored in the SSA.

ERESUME Memory Parameter Semantics

TCS
Enclave read/write access

The instruction faults if any of the following:

Address in RBX is not properly aligned.	Any TCS.FLAGS's must-be-zero bit is not zero.
TCS pointed to by RBX is not valid or available or locked.	Current 32/64 mode does not match the enclave mode in SECS.ATTRIBUTES.MODE64.
The SECS is in use by another enclave.	Either of TCS-specified FS and GS segment is not a subset of the current DS segment.
Any one of DS, ES, CS, SS is not zero.	If XSAVE available, CR4.OSXSAVE = 0, but SECS.ATTRIBUTES.XFRM ≠ 3.
CR4.OSFXSR ≠ 1.	If CR4.OSXSAVE = 1, SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
Offsets 520-535 of the XSAVE area not 0.	The bit vector stored at offset 512 of the XSAVE area must be a subset of SECS.ATTRIBUTES.XFRM.
The SSA frame is not valid or in use.	

The following operations are performed by ERESUME:

- RSP and RBP are saved in the current SSA frame on EENTER and are automatically restored on EEXIT or an asynchronous exit due to any Interrupt event.
- The AEP contained in RCX is stored into the TCS for use by AEXs.FS and GS (including hidden portions) are saved and new values are constructed using TCS.OFSBASE/GSBASE (32 and 64-bit mode) and TCS.OFSLIMIT/GSLIMIT (32-bit mode only). The resulting segments must be a subset of the DS segment.
- If CR4.OSXSAVE == 1, XCR0 is saved and replaced by SECS.ATTRIBUTES.XFRM. The effect of RFLAGS.TF depends on whether the enclave entry is opt-in or opt-out (see Section 39.1.2):
  - — On opt-out entry, TF is saved and cleared (it is restored on EEXIT or AEX). Any attempt to set TF via a POPF instruction while inside the enclave clears TF (see Section 39.2.5).
  - — On opt-in entry, a single-step debug exception is pended on the instruction boundary immediately after EENTER (see Section 39.2.3).
- All code breakpoints that do not overlap with ELRANGE are also suppressed. If the entry is an opt-out entry, all code and data breakpoints that overlap with the ELRANGE are suppressed.

- On opt-out entry, a number of performance monitoring counters and behaviors are modified or suppressed (see Section 39.2.3):
  - All performance monitoring activity on the current thread is suppressed except for incrementing and firing of FIXED_CTR1 and FIXED_CTR2.
  - PEBS is suppressed.
  - AnyThread counting on other threads is demoted to MyThread mode and IA32_PERF_GLOBAL_STATUS[60] on that thread is set.
  - If the opt-out entry on a hardware thread results in suppression of any performance monitoring, then the processor sets IA32_PERF_GLOBAL_STATUS[60] and IA32_PERF_GLOBAL_STATUS[63].

Concurrency Restrictions

Table 37-72. Base Concurrency Restrictions of ERESUME

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ERESUME	TCS [DS:RBX]	Shared	#GP	

Table 37-73. Additional Concurrency Restrictions of ERESUME

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ERESUME	TCS [DS:RBX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ERESUME Operational Flow

Name	Type	Size	Description
TMP_FSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_GSBASE	Effective Address	32/64	Proposed base address for FS segment.
TMP_FSLIMIT	Effective Address	32/64	Highest legal address in proposed FS segment.
TMP_GSLIMIT	Effective Address	32/64	Highest legal address in proposed GS segment.
TMP_TARGET	Effective Address	32/64	Address of first instruction inside enclave at which execution is to resume.
TMP_SECS	Effective Address	32/64	Physical address of SECS for this enclave.
TMP_SSA	Effective Address	32/64	Address of current SSA frame.
TMP_XSIZE	integer	64	Size of XSAVE area based on SECS.ATTRIBUTES.XFRM.
TMP_SSA_PAGE	Effective Address	32/64	Pointer used to iterate over the SSA pages in the current frame.
TMP_GPR	Effective Address	32/64	Address of the GPR area within the current SSA frame.
TMP_BRANCH_RECORD	LBR Record		From/to addresses to be pushed onto the LBR stack.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Make sure DS is usable, expand up *)
IF (TMP_MODE64 = 0 and (DS not usable or ( ( DS[S] = 1) and (DS[bit 11] = 0) and DS[bit 10] = 1) ) )
    THEN #GP(0); FI;
```

```
(* Check that CS, SS, DS, ES.base is 0 *)
IF (TMP_MODE64 = 0)
    THEN
        IF(CS.base ≠ 0 or DS.base ≠ 0) #GP(0); FI;
        IF(ES usable and ES.base ≠ 0) #GP(0); FI;
        IF(SS usable and SS.base ≠ 0) #GP(0); FI;
        IF(SS usable and SS.B = 0) #GP(0); FI;
FI;

IF (DS:RBX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

(* Check AEP is canonical*)
IF (TMP_MODE64 = 1 and (CS:RCX is not canonical) )
    THEN #GP(0); FI;

(* Check concurrency of TCS operation*)
IF (Other Intel SGX instructions is operating on TCS)
    THEN #GP(0); FI;

(* TCS verification *)
IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

IF ((EPCM(DS:RBX).PENDING = 1) or (EPCM(DS:RBX).MODIFIED = 1))
    THEN #PF(DS:RBX); FI;

IF ( (EPCM(DS:RBX).ENCLAVEADDRESS ≠ DS:RBX) or (EPCM(DS:RBX).PT ≠ PT_TCS) )
    THEN #PF(DS:RBX); FI;

IF ( (DS:RBX).OSSA is not 4KByte Aligned)
    THEN #GP(0); FI;

(* Check proposed FS and GS *)
IF ( ( (DS:RBX).OFSBASE is not 4KByte Aligned) or ( (DS:RBX).OGSBASE is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* Get the SECS for the enclave in which the TCS resides *)
TMP_SECS := Address of SECS for TCS;

(* Make sure that the FLAGS field in the TCS does not have any reserved bits set *)
IF ( ( (DS:RBX).FLAGS & FFFFFFFFFFFFFFFEH) ≠ 0)
    THEN #GP(0); FI;

(* SECS must exist and enclave must have previously been EINITted *)
IF (the enclave is not already initialized)
    THEN #GP(0); FI;
```

```
(* make sure the logical processor's operating mode matches the enclave *)
IF ( (TMP_MODE64 ≠ TMP_SECS.ATTRIBUTES.MODE64BIT) )
    THEN #GP(0); FI;

IF (CR4.OSFXSR = 0)
    THEN #GP(0); FI;

(* Check for legal values of SECS.ATTRIBUTES.XFRM *)
IF (CR4.OSXSAVE = 0)
    THEN
        IF (TMP_SECS.ATTRIBUTES.XFRM ≠ 03H) THEN #GP(0); FI;
    ELSE
        IF ( (TMP_SECS.ATTRIBUTES.XFRM & XCR0) ≠ TMP_SECS.ATTRIBUTES.XFRM) THEN #GP(0); FI;
FI;

(* Make sure the SSA contains at least one active frame *)
IF ( (DS:RBX).CSSA = 0)
    THEN #GP(0); FI;

(* Compute linear address of SSA frame *)
TMP_SSA := (DS:RBX).OSSA + TMP_SECS.BASEADDR + 4096 * TMP_SECS.SSAFRAMESIZE * ( (DS:RBX).CSSA - 1);
TMP_XSIZE := compute_XSAVE_frame_size(TMP_SECS.ATTRIBUTES.XFRM);

FOR EACH TMP_SSA_PAGE = TMP_SSA to TMP_SSA + TMP_XSIZE
    (* Check page is read/write accessible *)
    Check that DS:TMP_SSA_PAGE is read/write accessible;
    If a fault occurs, release locks, abort and deliver that fault;
    IF (DS:TMP_SSA_PAGE does not resolve to EPC page)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).VALID = 0)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF (EPCM(DS:TMP_SSA_PAGE).BLOCKED = 1)
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ((EPCM(DS:TMP_SSA_PAGE).PENDING = 1) or (EPCM(DS:TMP_SSA_PAGE_.MODIFIED = 1))
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    IF ( ( EPCM(DS:TMP_SSA_PAGE).ENCLAVEADDRESS ≠ DS:TMPSSA_PAGE) or (EPCM(DS:TMP_SSA_PAGE).PT ≠ PT_REG) or
        (EPCM(DS:TMP_SSA_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
        (EPCM(DS:TMP_SSA_PAGE).R = 0) or (EPCM(DS:TMP_SSA_PAGE).W = 0) )
        THEN #PF(DS:TMP_SSA_PAGE); FI;
    CR_XSAVE_PAGE_n := Physical_Address(DS:TMP_SSA_PAGE);
ENDFOR

(* Compute address of GPR area*)
TMP_GPR := TMP_SSA + 4096 * DS:TMP_SECS.SSAFRAMESIZE - sizeof(GPRSGX_AREA);
Check that DS:TMP_SSA_PAGE is read/write accessible;
If a fault occurs, release locks, abort and deliver that fault;
IF (DS:TMP_GPR does not resolve to EPC page)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).VALID = 0)
    THEN #PF(DS:TMP_GPR); FI;
IF (EPCM(DS:TMP_GPR).BLOCKED = 1)
    THEN #PF(DS:TMP_GPR); FI;
IF ((EPCM(DS:TMP_GPR).PENDING = 1) or (EPCM(DS:TMP_GPR).MODIFIED = 1))
    THEN #PF(DS:TMP_GPR); FI;
```

```
IF ( ( EPCM(DS:TMP_GPR).ENCLAVEADDRESS ≠ DS:TMP_GPR) or (EPCM(DS:TMP_GPR).PT ≠ PT_REG) or
    (EPCM(DS:TMP_GPR).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS) or
    (EPCM(DS:TMP_GPR).R = 0) or (EPCM(DS:TMP_GPR).W = 0) )
    THEN #PF(DS:TMP_GPR); FI;

IF (TMP_MODE64 = 0)
    THEN
        IF (TMP_GPR + (GPR_SIZE -1) is not in DS segment) THEN #GP(0); FI;
FI;

CR_GPR_PA := Physical_Address (DS: TMP_GPR);

TMP_TARGET := (DS:TMP_GPR).RIP;
IF (TMP_MODE64 = 1)
    THEN
        IF (TMP_TARGET is not canonical) THEN #GP(0); FI;
    ELSE
        IF (TMP_TARGET > CS limit) THEN #GP(0); FI;
FI;

(* Check proposed FS/GS segments fall within DS *)
IF (TMP_MODE64 = 0)
    THEN
        TMP_FSBASE := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR;
        TMP_FSLIMIT := (DS:RBX).OFSBASE + TMP_SECS.BASEADDR + (DS:RBX).FSLIMIT;
        TMP_GSBASE := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR;
        TMP_GSLIMIT := (DS:RBX).OGSBASE + TMP_SECS.BASEADDR + (DS:RBX).GSLIMIT;
        (* if FS wrap-around, make sure DS has no holes*)
        IF (TMP_FSLIMIT < TMP_FSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_FSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
        (* if GS wrap-around, make sure DS has no holes*)
        IF (TMP_GSLIMIT < TMP_GSBASE)
            THEN
                IF (DS.limit < 4GB) THEN #GP(0); FI;
            ELSE
                IF (TMP_GSLIMIT > DS.limit) THEN #GP(0); FI;
        FI;
    ELSE
        TMP_FSBASE := DS:TMP_GPR.FSBASE;
        TMP_GSBASE := DS:TMP_GPR.GSBASE;
        IF ( (TMP_FSBASE is not canonical) or (TMP_GSBASE is not canonical))
            THEN #GP(0); FI;
FI;

(* Ensure the enclave is not already active and this thread is the only one using the TCS*)
IF (DS:RBX.STATE = ACTIVE))
    THEN #GP(0); FI;

TMP_IA32_U_CET := 0
TMP_SSP := 0
```

```
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        IF ( CR4.CET = 0 )
            THEN
                (* If part does not support CET or CET has not been enabled and enclave requires CET then fail *)
                IF ( TMP_SECS.CET_ATTRIBUTES ≠ 0 OR TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0 ) #GP(0); FI;
        FI;
        (* If indirect branch tracking or shadow stacks enabled but CET state save area is not 16B aligned then fail ERESUME *)
        IF ( TMP_SECS.CET_ATTRIBUTES.SH_STK_EN = 1 OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN = 1 )
            THEN
                IF (DS:RBX.OCETSSA is not 16B aligned) #GP(0); FI;
        FI;

IF (TMP_SECS.CET_ATTRIBUTES.SH_STK_EN OR TMP_SECS.CET_ATTRIBUTES.ENDBR_EN)
    THEN
        (* Setup CET state from SECS, note tracker goes to IDLE *)
        TMP_IA32_U_CET = TMP_SECS.CET_ATTRIBUTES;
        IF (TMP_IA32_U_CET.LEG_IW_EN = 1 AND TMP_IA32_U_CET.ENDBR_EN = 1 )
            THEN
                TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.BASEADDR;
                TMP_IA32_U_CET := TMP_IA32_U_CET + TMP_SECS.CET_LEG_BITMAP_BASE;
        FI;

        (* Compute linear address of what will become new CET state save area and cache its PA *)
        TMP_CET_SAVE_AREA = DS:RBX.OCETSSA + TMP_SECS.BASEADDR + (DS:RBX.CSSA - 1) * 16
        TMP_CET_SAVE_PAGE = TMP_CET_SAVE_AREA & ~0xFFF;

        Check the TMP_CET_SAVE_PAGE page is read/write accessible
        If fault occurs release locks, abort and deliver fault

        (* read the EPCM VALID, PENDING, MODIFIED, BLOCKED and PT fields atomically *)
        IF ((DS:TMP_CET_SAVE_PAGE Does NOT RESOLVE TO EPC PAGE) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).VALID = 0) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).PENDING = 1) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).MODIFIED = 1) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).BLOCKED = 1) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).R = 0) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).W = 0) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVEADDRESS ≠ DS:TMP_CET_SAVE_PAGE) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).PT ≠ PT_SS_REST) OR
         (EPCM(DS:TMP_CET_SAVE_PAGE).ENCLAVESECS ≠ EPCM(DS:RBX).ENCLAVESECS))
            THEN
                #PF(DS:TMP_CET_SAVE_PAGE);
        FI;

        CR_CET_SAVE_AREA_PA := Physical address(DS:TMP_CET_SAVE_AREA)

        TMP_SSP = CR_CET_SAVE_AREA_PA.SSP
        TMP_IA32_U_CET.TRACKER = CR_CET_SAVE_AREA_PA.TRACKER;
        TMP_IA32_U_CET.SUPPRESS = CR_CET_SAVE_AREA_PA.SUPPRESS;

        IF ( (TMP_MODE64 = 1 AND TMP_SSP is not canonical) OR
         (TMP_MODE64 = 0 AND (TMP_SSP & 0xFFFFFFFF00000000) ≠ 0) OR
```

```
            (TMP_SSP is not 4 byte aligned) OR
            (TMP_IA32_U_CET.TRACKER = WAIT_FOR_ENDBRANCH AND TMP_IA32_U_CET.SUPPRESS = 1) OR
            (CR_CET_SAVE_AREA_PA.Reserved ≠ 0) ) #GP(0); FI;
          FI;
FI;

(* SECS.ATTRIBUTES.XFRM selects the features to be saved. *)
(* CR_XSAVE_PAGE_n: A list of 1 or more physical address of pages that contain the XSAVE area. *)
XRSTOR(TMP_MODE64, SECS.ATTRIBUTES.XFRM, CR_XSAVE_PAGE_n);

IF (XRSTOR failed with #GP)
    THEN
        DS:RBX.STATE := INACTIVE;
        #GP(0);
FI;

CR_ENCLAVE_MODE := 1;
CR_ACTIVE_SECS := TMP_SECS;
CR_ELRANGE := (TMP_SECS.BASEADDR, TMP_SECS.SIZE);

(* Save sate for possible AEXs *)
CR_TCS_PA := Physical_Address (DS:RBX);
CR_TCS_LA := RBX;
CR_TCS_LA.AEP := RCX;

(* Save the hidden portions of FS and GS *)
CR_SAVE_FS_selector := FS.selector;
CR_SAVE_FS_base := FS.base;
CR_SAVE_FS_limit := FS.limit;
CR_SAVE_FS_access_rights := FS.access_rights;
CR_SAVE_GS_selector := GS.selector;
CR_SAVE_GS_base := GS.base;
CR_SAVE_GS_limit := GS.limit;
CR_SAVE_GS_access_rights := GS.access_rights;

RIP := TMP_TARGET;

Restore_GPRs from DS:TMP_GPR;

(*Restore the RFLAGS values from SSA*)
RFLAGS.CF := DS:TMP_GPR.RFLAGS.CF;
RFLAGS.PF := DS:TMP_GPR.RFLAGS.PF;
RFLAGS.AF := DS:TMP_GPR.RFLAGS.AF;
RFLAGS.ZF := DS:TMP_GPR.RFLAGS.ZF;
RFLAGS.SF := DS:TMP_GPR.RFLAGS.SF;
RFLAGS.DF := DS:TMP_GPR.RFLAGS.DF;
RFLAGS.OF := DS:TMP_GPR.RFLAGS.OF;
RFLAGS.NT := DS:TMP_GPR.RFLAGS.NT;
RFLAGS.AC := DS:TMP_GPR.RFLAGS.AC;
RFLAGS.ID := DS:TMP_GPR.RFLAGS.ID;
RFLAGS.RF := DS:TMP_GPR.RFLAGS.RF;
RFLAGS.VM := 0;
IF (RFLAGS.IOPL = 3)
    THEN RFLAGS.IF := DS:TMP_GPR.RFLAGS.IF; FI;
```

```
IF (TCS.FLAGS.OPTIN = 0)
    THEN RFLAGS.TF := 0; FI;

(* If XSAVE is enabled, save XCR0 and replace it with SECS.ATTRIBUTES.XFRM*)
IF (CR4.OSXSAVE = 1)
    CR_SAVE_XCR0 := XCR0;
    XCR0 := TMP_SECS.ATTRIBUTES.XFRM;
FI;

(* Pop the SSA stack*)
(DS:RBX).CSSA := (DS:RBX).CSSA -1;

(* Do the FS/GS swap *)
FS.base := TMP_FSBASE;
FS.limit := DS:RBX.FSLIMIT;
FS.type := 0001b;
FS.W := DS.W;
FS.S := 1;
FS.DPL := DS.DPL;
FS.G := 1;
FS.B := 1;
FS.P := 1;
FS.AVL := DS.AVL;
FS.L := DS.L;
FS.unusable := 0;
FS.selector := 0BH;

GS.base := TMP_GSBASE;
GS.limit := DS:RBX.GSLIMIT;
GS.type := 0001b;
GS.W := DS.W;
GS.S := 1;
GS.DPL := DS.DPL;
GS.G := 1;
GS.B := 1;
GS.P := 1;
GS.AVL := DS.AVL;
GS.L := DS.L;
GS.unusable := 0;
GS.selector := 0BH;

CR_DBGOPTIN := TCS.FLAGS.DBGOPTIN;
Suppress all code breakpoints that are outside ELRANGE;

IF (CR_DBGOPTIN = 0)
    THEN
        Suppress all code breakpoints that overlap with ELRANGE;
        CR_SAVE_TF := RFLAGS.TF;
        RFLAGS.TF := 0;
        Suppress any MTF VM exits during execution of the enclave;
        Clear all pending debug exceptions;
        Clear any pending MTF VM exit;
    ELSE
```

```
        Clear all pending debug exceptions;
        Clear pending MTF VM exits;
FI;

IF ((CPUID.(EAX=7H, ECX=0):EDX[CET_IBT] = 1) OR (CPUID.(EAX=7, ECX=0):ECX[CET_SS] = 1)
    THEN
        (* Save enclosing application CET state into save registers *)
        CR_SAVE_IA32_U_CET := IA32_U_CET
        (* Setup enclave CET state *)
        IF CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1
            THEN
                CR_SAVE_SSP := SSP
                SSP := TMP_SSP;
        FI;
        IA32_U_CET := TMP_IA32_U_CET;
FI;

(* Assure consistent translations *)
Flush_linear_context;
Clear_Monitor_FSM;
Allow_front_end_to_begin_fetch_at_new_RIP;
```

Flags Affected

RFLAGS.TF is cleared on opt-out entry

Protected Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If the enclave is not initialized.
	If the thread is not in the INACTIVE state.
	If CS, DS, ES or SS bases are not all zero.
	If executed in enclave mode.
	If part or all of the FS or GS segment specified by TCS is outside the DS segment.
	If any reserved field in the TCS FLAG is set.
	If the target address is not within the CS segment.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
#PF(error code)	If a page fault occurs in accessing memory.
	If DS:RBX does not point to a valid TCS.
	If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

64-Bit Mode Exceptions

#GP(0)	If DS:RBX is not page aligned.
	If the enclave is not initialized.
	If the thread is not in the INACTIVE state.
	If CS, DS, ES or SS bases are not all zero.
	If executed in enclave mode.
	If part or all of the FS or GS segment specified by TCS is outside the DS segment.
	If any reserved field in the TCS FLAG is set.

	If the target address is not canonical.
	If CR4.OSFXSR = 0.
	If CR4.OSXSAVE = 0 and SECS.ATTRIBUTES.XFRM ≠ 3.
	If CR4.OSXSAVE = 1and SECS.ATTRIBUTES.XFRM is not a subset of XCR0.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If DS:RBX does not point to a valid TCS.
	If one or more pages of the current SSA frame are not readable/writable, or do not resolve to a valid PT_REG EPC page.

37.5 INTEL® SGX VIRTUALIZATION LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLV instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional implicit registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of each implicit register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EDECVIRTCHILD—Decrement VIRTCHILDCNT in SECS

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLV[EDECVIRTCHILD]	IR	V/V	EAX[5]	This leaf function decrements the SECS VIRTCHILDCNT field.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EDECVIRTCHILD (In)	Return error code (Out)	Address of an enclave page (In)	Address of an SECS page (In)

Description

This instruction decrements the SECS VIRTCHILDCNT field. This instruction can only be executed when current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create linear address. Segment override is not supported.

EDECVIRTCHILD Memory Parameter Semantics

EPCPAGE	SECS
Read/Write access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EDECVIRTCHILD Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A page fault occurs in accessing memory operands.
DS segment is unusable (32b mode).	RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
A memory address is in a non-canonical form (64b mode).	RCX does not refer to an SECS page.
A memory operand is not properly aligned.	RBX does not refer to an enclave page associated with SECS referenced in RCX.

Concurrency Restrictions

Table 37-74. Base Concurrency Restrictions of EDECVIRTCHILD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EDECVIRTCHILD	Target [DS:RBX]	Shared	SGX_EPC_PAGE_ CONFLICT	
	SECS [DS:RCX]	Concurrent		

Table 37-75. Additional Concurrency Restrictions of EDECVIRTCHILD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EDECVIRTCHILD	Target [DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EDECVIRTCHILD Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.
TMP_VIRTCHILDCNT	Integer	64	Number of virtual child pages.

EDECVIRTCHILD Return Value in RAX

Error	Value	Description
No Error	0	EDECVIRTCHILD Successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.
SGX_INVALID_COUNTER		Attempt to decrement counter that is already zero.

```
(* check alignment of DS:RBX *)
IF (DS:RBX is not 4K aligned) THEN
   #GP(0); FI;

(* check DS:RBX is an linear address of an EPC page *)
IF (DS:RBX does not resolve within an EPC) THEN
   #PF(DS:RBX, PFEC.SGX); FI;

(* check DS:RCX is an linear address of an EPC page *)
IF (DS:RCX does not resolve within an EPC) THEN
   #PF(DS:RCX, PFEC.SGX); FI;

(* Check the EPCPAGE for concurrency *)
IF (EPCPAGE is being modified) THEN
   RFLAGS.ZF = 1;
   RAX = SGX_EPC_PAGE_CONFLICT;
   goto DONE;
FI;

(* check that the EPC page is valid *)
IF (EPCM(DS:RBX).VALID = 0) THEN
   #PF(DS:RBX, PFEC.SGX); FI;

(* check that the EPC page has the correct type and that the back pointer matches the pointer passed as the pointer to parent *)
IF ((EPCM(DS:RBX).PAGE_TYPE = PT_REG) or
   (EPCM(DS:RBX).PAGE_TYPE = PT_TCS) or
```

```
    (EPCM(DS:RBX).PAGE_TYPE = PT_TRIM) or
    (EPCM(DS:RBX).PAGE_TYPE = PT_SS_FIRST) or
    (EPCM(DS:RBX).PAGE_TYPE = PT_SS_REST))
      THEN
    (* get the SECS of DS:RBX *)
    TMP_SECS := Address of SECS for (DS:RBX);
ELSE IF (EPCM(DS:RBX).PAGE_TYPE = PT_SECS) THEN
    (* get the physical address of DS:RBX *)
    TMP_SECS := Physical_Address(DS:RBX);
ELSE
    (* EDECVIRTCHILD called on page of incorrect type *)
    #PF(DS:RBX, PFEC.SGX); FI;

IF (TMP_SECS ≠ Physical_Address(DS:RCX)) THEN
    #GP(0); FI;

(* Atomically decrement virtchild counter and check for underflow *)
Locked_Decrement(SECS(TMP_SECS).VIRTCHILDCNT);
IF (There was an underflow) THEN
    Locked_Increment(SECS(TMP_SECS).VIRTCHILDCNT);
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_COUNTER;
    goto DONE;
FI;

RFLAGS.ZF := 0;
RAX := 0;

DONE:
(* clear flags *)
RFLAGS.CF := 0;
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;
```

Flags Affected

ZF is set if EDECVIRTCHILD fails due to concurrent operation with another SGX instruction, or if there is a VIRTCHILDCNT underflow. Otherwise cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If DS segment is unusable.
	If a memory operand is not properly aligned.
	RBX does not refer to an enclave page associated with SECS referenced in RCX.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
	If RCX does not refer to an SECS page.

64-Bit Mode Exceptions

#GP(0)	If a memory address is in a non-canonical form. If a memory operand is not properly aligned. RBX does not refer to an enclave page associated with SECS referenced in RCX.
#PF(error code)	If a page fault occurs in accessing memory operands. If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS). If RCX does not refer to an SECS page.

EINCVIRTCHILD—Increment VIRTCHILDCNT in SECS

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLV[EINCVIRTCHILD]	IR	V/V	EAX[5]	This leaf function increments the SECS VIRTCHILDCNT field.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EINCVIRTCHILD (In)	Return error code (Out)	Address of an enclave page (In)	Address of an SECS page (In)

Description

This instruction increments the SECS VIRTCHILDCNT field. This instruction can only be executed when the current privilege level is 0.

The content of RCX is an effective address of an EPC page. The DS segment is used to create a linear address. Segment override is not supported.

EINCVIRTCHILD Memory Parameter Semantics

EPCPAGE	SECS
Read/Write access permitted by Non Enclave	Read access permitted by Enclave

The instruction faults if any of the following:

EINCVIRTCHILD Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A page fault occurs in accessing memory operands.
DS segment is unusable (32b mode).	RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
A memory address is in a non-canonical form (64b mode).	RCX does not refer to an SECS page.
A memory operand is not properly aligned.	RBX does not refer to an enclave page associated with SECS referenced in RCX.

Concurrency Restrictions

Table 37-76. Base Concurrency Restrictions of EINCVIRTCHILD

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
EINCVIRTCHILD	Target [DS:RBX]	Shared	SGX_EPC_PAGE_ CONFLICT	
	SECS [DS:RCX]	Concurrent		

Table 37-77. Additional Concurrency Restrictions of EINCVIRTCHILD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EINCVIRTCHILD	Target [DS:RBX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EINCVIRTCHILD Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.

EINCVIRTCHILD Return Value in RAX

Error	Value	Description
No Error	0	EINCVIRTCHILD Successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.

```
(* check alignment of DS:RBX *)
IF (DS:RBX is not 4K aligned) THEN
   #GP(0); FI;

(* check DS:RBX is an linear address of an EPC page *)
IF (DS:RBX does not resolve within an EPC) THEN
   #PF(DS:RBX, PFEC.SGX); FI;

(* check DS:RCX is an linear address of an EPC page *)
IF (DS:RCX does not resolve within an EPC) THEN
   #PF(DS:RCX, PFEC.SGX); FI;

(* Check the EPCPAGE for concurrency *)
IF (EPCPAGE is being modified) THEN
   RFLAGS.ZF = 1;
   RAX = SGX_EPC_PAGE_CONFLICT;
   goto DONE;
FI;

(* check that the EPC page is valid *)
IF (EPCM(DS:RBX).VALID = 0) THEN
   #PF(DS:RBX, PFEC.SGX); FI;

(* check that the EPC page has the correct type and that the back pointer matches the pointer passed as the pointer to parent *)
IF ((EPCM(DS:RBX).PAGE_TYPE = PT_REG) or
   (EPCM(DS:RBX).PAGE_TYPE = PT_TCS) or
   (EPCM(DS:RBX).PAGE_TYPE = PT_TRIM) or
   (EPCM(DS:RBX).PAGE_TYPE = PT_SS_FIRST) or
   (EPCM(DS:RBX).PAGE_TYPE = PT_SS_REST))
```

```
    THEN
  (* get the SECS of DS:RBX *)
  TMP_SECS := Address of SECS for (DS:RBX);
ELSE IF (EPCM(DS:RBX).PAGE_TYPE = PT_SECS) THEN
  (* get the physical address of DS:RBX *)
  TMP_SECS := Physical_Address(DS:RBX);
ELSE
  (* EINCVIRTCHILD called on page of incorrect type *)
  #PF(DS:RBX, PFEC.SGX); FI;

IF (TMP_SECS ≠ Physical_Address(DS:RCX)) THEN
  #GP(0); FI;

(* Atomically increment virtchild counter *)
Locked_Increment(SECS(TMP_SECS).VIRTCHILDCNT);

RFLAGS.ZF := 0;
RAX := 0;

DONE:
(* clear flags *)
RFLAGS.CF := 0;
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;
```

Flags Affected

ZF is set if EINCVIRTCHILD fails due to concurrent operation with another SGX instruction; otherwise cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If DS segment is unusable.
	If a memory operand is not properly aligned.
	RBX does not refer to an enclave page associated with SECS referenced in RCX.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
	If RCX does not refer to an SECS page.

64-Bit Mode Exceptions

#GP(0)	If a memory address is in a non-canonical form.
	If a memory operand is not properly aligned.
	RBX does not refer to an enclave page associated with SECS referenced in RCX.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If RBX does not refer to an enclave page (REG, TCS, TRIM, SECS).
	If RCX does not refer to an SECS page.

ESETCONTEXT—Set the ENCLAVECONTEXT Field in SECS

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLV[ESETCONTEXT]	IR	V/V	EAX[5]	This leaf function sets the ENCLAVECONTEXT field in SECS.

Instruction Operand Encoding

Op/En	EAX		RCX	RDX
IR	ESETCONTEXT (In)	Return error code (Out)	Address of the destination EPC page (In, EA)	Context Value (In, EA)

Description

The ESETCONTEXT leaf overwrites the ENCLAVECONTEXT field in the SECS. ECREATE and ELD of an SECS set the ENCLAVECONTEXT field in the SECS to the address of the SECS (for access later in ERDINFO). The ESETCONTEXT instruction allows a VMM to overwrite the default context value if necessary, for example, if the VMM is emulating ECREATE or ELD on behalf of the guest.

The content of RCX is an effective address of the SECS page to be updated, RDX contains the address pointing to the value to be stored in the SECS. The DS segment is used to create linear address. Segment override is not supported.

The instruction fails if:

- The operand is not properly aligned.
- RCX does not refer to an SECS page.

ESETCONTEXT Memory Parameter Semantics

EPCPAGE	CONTEXT
Read access permitted by Enclave	Read/Write access permitted by Non Enclave

The instruction faults if any of the following:

ESETCONTEXT Faulting Conditions

A memory operand effective address is outside the DS segment limit (32b mode).	A memory operand is not properly aligned.
DS segment is unusable (32b mode).	A page fault occurs in accessing memory operands.
A memory address is in a non-canonical form (64b mode).	

Concurrency Restrictions

Table 37-78. Base Concurrency Restrictions of ESETCONTEXT

Leaf	Parameter	Base Concurrency Restrictions		
		Access	On Conflict	SGX_CONFLICT VM Exit Qualification
ESETCONTEXT	SECS [DS:RCX]	Shared	SGX_EPC_PAGE_ CONFLICT	

Table 37-79. Additional Concurrency Restrictions of ESETCONTEXT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK, ETRACKC	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ESETCONTEXT	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ESETCONTEXT Operational Flow

Name	Type	Size (bits)	Description
TMP_SECS	Physical Address	64	Physical address of the SECS of the page being modified.
TMP_CONTEXT	CONTEXT	64	Data Value of CONTEXT.

ESETCONTEXT Return Value in RAX

Error	Value	Description
No Error	0	ESETCONTEXT Successful.
SGX_EPC_PAGE_CONFLICT		Failure due to concurrent operation of another SGX instruction.

```
(* check alignment of the EPCPAGE (RCX) *)
IF (DS:RCX is not 4KByte Aligned) THEN
   #GP(0); FI;

 (* check that EPCPAGE (DS:RCX) is the address of an EPC page *)
IF (DS:RCX does not resolve within an EPC)THEN
   #PF(DS:RCX, PFEC.SGX); FI;

(* check alignment of the CONTEXT field (RDX) *)
IF (DS:RDX is not 8Byte Aligned) THEN
   #GP(0); FI;

 (* Load CONTEXT into local variable *)
TMP_CONTEXT := DS:RDX

(* Check the EPC page for concurrency *)
IF (EPC page is being modified) THEN
   RFLAGS.ZF := 1;
   RFLAGS.CF := 0;
   RAX := SGX_EPC_PAGE_CONFLICT;
   goto DONE;
FI;

(* check page validity *)
IF (EPCM(DS:RCX).VALID = 0) THEN
   #PF(DS:RCX, PFEC.SGX);
FI;

(* check EPC page is an SECS page *)
```

```
IF (EPCM(DS:RCX).PT is not PT_SECS) THEN
   #PF(DS:RCX, PFEC.SGX);
FI;

(* load the context value into SECS(DS:RCX).ENCLAVECONTEXT *)
SECS(DS:RCX).ENCLAVECONTEXT := TMP_CONTEXT;

RAX := 0;
RFLAGS.ZF := 0;

DONE:
(* clear flags *)
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

ZF is set if ESETCONTEXT fails due to concurrent operation with another SGX instruction; otherwise cleared. CF, PF, AF, OF and SF are cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If DS segment is unusable.
	If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If a memory address is in a non-canonical form.
	If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands.

26. Updates to Appendix A, Volume 3D

Change bars and green 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 chapter: Updated Section A.4, "VM-Exit Controls". Section A.4.1 renamed "Primary VM-Exit Controls" and added new Section A.4.2, "Secondary VM-Exit Controls". Update to Section A.10, "VPID and EPT Capabilities".

APPENDIX A
VMX CAPABILITY REPORTING FACILITY

The ability of a processor to support VMX operation and related instructions is indicated by CPUID.1:ECX.VMX[bit 5] = 1. A value 1 in this bit indicates support for VMX features.

Support for specific features detailed in Chapter 26 and other VMX chapters is determined by reading values from a set of capability MSRs. 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 RDMSR) 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 31.15 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.
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 27.2.5). 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 26.2.1.3).
- The values of bits 47:45 and bits 63:57 are reserved and are read as 0.

A.2 RESERVED CONTROLS AND DEFAULT SETTINGS

As noted in Chapter 26, "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 26.2.1).
- 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_CTLS MSR (index 481H) reports on the allowed settings of **most** of the pin-based VM-execution controls (see Section 24.6.1):

- 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_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 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_CTLS 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_CTLS 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_CTLS MSR. (The IA32_VMX_TRUE_PINBASED_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 pin-based VM-execution controls is contained in the IA32_VMX_TRUE_PINBASED_CTLS 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_CTLS MSR.

A.3.2 Primary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS MSR (index 482H) reports on the allowed settings of **most** of the primary processor-based VM-execution controls (see Section 24.6.2):

- 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_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 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_CTLS 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_CTLS MSR (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_CTLS MSR. (The IA32_VMX_TRUE_PROCBASED_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 processor-based VM-execution controls is contained in the IA32_VMX_TRUE_PROCBASED_CTLS MSR. Assuming that software knows that the default1 class of 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_CTLS MSR.

A.3.3 Secondary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS2 MSR (index 48BH) reports on the allowed settings of the secondary processor-based VM-execution controls (see Section 24.6.2). 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_CTLS2 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_CTLS MSR is 1).

A.3.4 Tertiary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS3 MSR (index 492H) reports on the allowed 1-settings of the tertiary processor-based VM-execution controls (see Section 24.6.2); 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_CTLS3 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_CTLS MSR is 1).

Notice that the organization of this MSR differs from that of IA32_VMX_PROCBASED_CTLS2 (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 24.7.1):

- 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 24.7.1); 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 24.8.1):

- 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_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 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_CTLS MSR (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_CTLS MSR (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_CTLS MSR. (The IA32_VMX_TRUE_ENTRY_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 VM-entry controls is contained in the IA32_VMX_TRUE_ENTRY_CTLS MSR. 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_CTLS MSR.

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 27.2 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 30); 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 31.15.6.3.
- 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 31.14.4).
- 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 24.11.2, 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 28.1) and extended page tables (EPT, Section 28.3):

- 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.
- 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 24.6.11.
- 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 30 and Section 28.4.3.1):
  - 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 28.3.5).
- If bit 22 is read as 1, the processor reports advanced VM-exit information for EPT violations (see Section 27.2.1). 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 28.3.3.2).
- Support for the INVVPID instruction (see Chapter 30 and Section 28.4.3.1):
  - 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 4.5.1.
- Bits 5:1, bit 7, 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_CTLS MSR is 1) and that support either the 1-setting of the "enable EPT" VM-execution control (only if bit 33 of the IA32_VMX_PROCBASED_CTLS2 MSR is 1) or the 1-setting of the "enable VPID" VM-execution control (only if bit 37 of the IA32_VMX_PROCBASED_CTLS2 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 24.6.14). 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.

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).

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_CTLS 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_CTLS2 MSR is 1).

27. Updates to Appendix B, Volume 3D

Change bars and green 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: Update to Table B-1, "Encoding for 16-Bit Control Fields (0000_00xx_xxxx_xxx0B)". Update to Table B-4, "Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) ". Update to Table B-6, "Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)". Minor naming update in Table B-8, "Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B)".

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 24.11.2 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 24.11.2, 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

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.

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
Guest GS selector	000000101B	0000080AH
Guest LDTR selector	000000110B	0000080CH

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest TR selector	000000111B	0000080EH
Guest interrupt status[1]	000001000B	00000810H
PML index[2]	000001001B	00000812H

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.

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 24.11.2, 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
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

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
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
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

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
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
ENCLV-exiting bitmap (full)[17]	000011011B	00002036H
ENCLV-exiting bitmap (high)[17]		00002037H
Hypervisor-managed linear-address translation pointer (HLATP; full)[18]	000100000B	00002040H
HLATP (high)[18]		00002041H
Secondary VM-exit controls (full)[19]	000100010B	00002044H
Secondary VM-exit controls (high)[19]		00002045H

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 "enable ENCLV exiting" VM-execution control.
18. This field exists only on processors that support the 1-setting of the "enable HLAT" VM-execution control.
19. This field exists only on processors that support the 1-setting of the "activate secondary controls" VM-exit control.

B.2.2 64-Bit Read-Only Data Field

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. There is only one such 64-bit field as given in Table B-5.(As with other 64-bit fields, this one

has two encodings.)

Table B-5. Encodings for 64-Bit Read-Only Data Field (0010_01xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest-physical address (full)[1]	000000000B	00002400H
Guest-physical address (high)[1]		00002401H

NOTES:
1. This field exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.

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
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

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.

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

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.

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 24.11.2, 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
VM-entry interruption-information field	000001011B	00004016H
VM-entry exception error code	000001100B	00004018H
VM-entry instruction length	000001101B	0000401AH
TPR threshold[1]	000001110B	0000401CH
Secondary processor-based VM-execution controls[2]	000001111b	0000401EH
PLE_Gap[3]	000010000b	00004020H
PLE_Window[3]	000010001b	00004022H

NOTES:

1. This field exists only on processors that support the 1-setting of the "use TPR shadow" VM-execution control.
2. This field exists only on processors that support the 1-setting of the "activate secondary controls" VM-execution control.
3. This field exists only on processors that support the 1-setting of the "PAUSE-loop exiting" 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
VM-exit interruption information	000000010B	00004404H
VM-exit interruption error code	000000011B	00004406H
IDT-vectoring information field	000000100B	00004408H
IDT-vectoring error code	000000101B	0000440AH
VM-exit instruction length	000000110B	0000440CH

Table B-9. Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
VM-exit instruction information	000000111B	0000440EH

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
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 24.11.2, 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
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

Table B-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
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
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.

28. Updates to Chapter 1, Volume 4

Change bars show changes to Chapter 1 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4:* Model-Specific Registers.

Changes to this chapter: Updated section 1.1 "Intel® 64 and IA-32 Processors Covered in this Manual" to add the 12th generation Intel® Core™ processor. Additional minor corrections in chapter.

CHAPTER 1
ABOUT THIS MANUAL

The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers* (order number 335592) is part of a set that describes the architecture and programming environment of Intel® 64 and IA-32 architecture processors. Other volumes in this set are:

- *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*: *Basic Architecture* (order number 253665).
- *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D: Instruction Set Reference* (order numbers 253666, 253667, 326018 and 334569).
- The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A, 3B, 3C & 3D: System Programming Guide* (order numbers 253668, 253669, 326019 and 332831).

The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1,* describes the basic architecture and programming environment of Intel 64 and IA-32 processors. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D,* describe the instruction set of the processor and the opcode structure. These volumes apply to application programmers and to programmers who write operating systems or executives. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A, 3B, 3C & 3D*, describe the operating-system support environment of Intel 64 and IA-32 processors. These volumes target operating-system and BIOS designers. In addition, *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*, and *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* address the programming environment for classes of software that host operating systems. The *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4,* describes the model-specific registers of Intel 64 and IA-32 processors.

1.1 INTEL® 64 AND IA-32 PROCESSORS COVERED IN THIS MANUAL

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™2 Duo processor
- Intel® Core™2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series
- Intel® Core™2 Extreme processor X7000 and X6800 series
- Intel® Core™2 Extreme QX6000 series
- Intel® Xeon® processor 7100 series

- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Core™2 Extreme QX9000 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel Atom® processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are built from 45 nm and 32 nm processes.
- Intel® Core™ i7 processor
- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Core™ i7-3930K processor
- 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series
- Intel® Xeon® processor E3-1200 product family
- Intel® Xeon® processor E5-2400/1400 product family
- Intel® Xeon® processor E5-4600/2600/1600 product family
- 3rd generation Intel® Core™ processors
- Intel® Xeon® processor E3-1200 v2 product family
- Intel® Xeon® processor E5-2400/1400 v2 product families
- Intel® Xeon® processor E5-4600/2600/1600 v2 product families
- Intel® Xeon® processor E7-8800/4800/2800 v2 product families
- 4th generation Intel® Core™ processors
- The Intel® Core™ M processor family
- Intel® Core™ i7-59xx Processor Extreme Edition
- Intel® Core™ i7-49xx Processor Extreme Edition
- Intel® Xeon® processor E3-1200 v3 product family
- Intel® Xeon® processor E5-2600/1600 v3 product families
- 5th generation Intel® Core™ processors
- Intel® Xeon® processor D-1500 product family
- Intel® Xeon® processor E5 v4 family
- Intel Atom® processor X7-Z8000 and X5-Z8000 series
- Intel Atom® processor Z3400 series
- Intel Atom® processor Z3500 series
- 6th generation Intel® Core™ processors
- Intel® Xeon® processor E3-1500m v5 product family
- 7th generation Intel® Core™ processors
- Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series
- Intel® Xeon® Processor Scalable Family
- 8th generation Intel® Core™ processors
- Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series
- Intel® Xeon® E processors
- 9th generation Intel® Core™ processors
- 2nd generation Intel® Xeon® Processor Scalable Family

- 10th generation Intel® Core™ processors
- 11th generation Intel® Core™ processors
- 3rd generation Intel® Xeon® Processor Scalable Family
- 12th generation Intel® Core™ processors

P6 family processors are IA-32 processors based on the P6 family microarchitecture. This includes the Pentium® Pro, Pentium® II, Pentium® III, and Pentium® III Xeon® processors.

The Pentium® 4, Pentium® D, and Pentium® processor Extreme Editions are based on the Intel NetBurst® microarchitecture. Most early Intel® Xeon® processors are based on the Intel NetBurst® microarchitecture. Intel Xeon processor 5000, 7100 series are based on the Intel NetBurst® microarchitecture.

The Intel® Core™ Duo, Intel® Core™ Solo and dual-core Intel® Xeon® processor LV are based on an improved Pentium® M processor microarchitecture.

The Intel® Xeon® processor 3000, 3200, 5100, 5300, 7200, and 7300 series, Intel® Pentium® dual-core, Intel® Core™2 Duo, Intel® Core™2 Quad, and Intel® Core™2 Extreme processors are based on Intel® Core™ microarchitecture.

The Intel® Xeon® processor 5200, 5400, 7400 series, Intel® Core™2 Quad processor Q9000 series, and Intel® Core™2 Extreme processors QX9000, X9000 series, Intel® Core™2 processor E8000 series are based on Enhanced Intel® Core™ microarchitecture.

The Intel Atom® processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are based on the Intel Atom® microarchitecture and supports Intel 64 architecture.

P6 family, Pentium® M, Intel® Core™ Solo, Intel® Core™ Duo processors, dual-core Intel® Xeon® processor LV, and early generations of Pentium 4 and Intel Xeon processors support IA-32 architecture. The Intel® Atom™ processor Z5xx series support IA-32 architecture.

The Intel® Xeon® processor 3000, 3200, 5000, 5100, 5200, 5300, 5400, 7100, 7200, 7300, 7400 series, Intel® Core™2 Duo, Intel® Core™2 Extreme, Intel® Core™2 Quad processors, Pentium® D processors, Pentium® Dual-Core processor, newer generations of Pentium 4 and Intel Xeon processor family support Intel® 64 architecture.

The Intel® Core™ i7 processor and Intel® Xeon® processor 3400, 5500, 7500 series are based on 45 nm Nehalem microarchitecture. Westmere microarchitecture is a 32 nm version of the Nehalem microarchitecture. Intel® Xeon® processor 5600 series, Intel Xeon processor E7 and various Intel Core i7, i5, i3 processors are based on the Westmere microarchitecture. These processors support Intel 64 architecture.

The Intel® Xeon® processor E5 family, Intel® Xeon® processor E3-1200 family, Intel® Xeon® processor E7-8800/4800/2800 product families, Intel® Core™ i7-3930K processor, and 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Sandy Bridge microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E7-8800/4800/2800 v2 product families, Intel® Xeon® processor E3-1200 v2 product family and 3rd generation Intel® Core™ processors are based on the Ivy Bridge microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E5-4600/2600/1600 v2 product families, Intel® Xeon® processor E5-2400/1400 v2 product families and Intel® Core™ i7-49xx Processor Extreme Edition are based on the Ivy Bridge-E microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E3-1200 v3 product family and 4th Generation Intel® Core™ processors are based on the Haswell microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E5-2600/1600 v3 product families and the Intel® Core™ i7-59xx Processor Extreme Edition are based on the Haswell-E microarchitecture and support Intel 64 architecture.

The Intel Atom® processor Z8000 series is based on the Airmont microarchitecture.

The Intel Atom® processor Z3400 series and the Intel Atom® processor Z3500 series are based on the Silvermont microarchitecture.

The Intel® Core™ M processor family, 5th generation Intel® Core™ processors, Intel® Xeon® processor D-1500 product family and the Intel® Xeon® processor E5 v4 family are based on the Broadwell microarchitecture and support Intel 64 architecture.

The Intel® Xeon® Processor Scalable Family, Intel® Xeon® processor E3-1500m v5 product family and 6th generation Intel® Core™ processors are based on the Skylake microarchitecture and support Intel 64 architecture.

The 7th generation Intel® Core™ processors are based on the Kaby Lake microarchitecture and support Intel 64 architecture.

The Intel Atom® processor C series, the Intel Atom® processor X series, the Intel® Pentium® processor J series, the Intel® Celeron® processor J series, and the Intel® Celeron® processor N series are based on the Goldmont microarchitecture.

The Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series is based on the Knights Landing microarchitecture and supports Intel 64 architecture.

The Intel® Pentium® Silver processor series, the Intel® Celeron® processor J series, and the Intel® Celeron® processor N series are based on the Goldmont Plus microarchitecture.

The 8th generation Intel® Core™ processors, 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on the Coffee Lake microarchitecture and support Intel 64 architecture.

The Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series is based on the Knights Mill microarchitecture and supports Intel 64 architecture.

The 2nd generation Intel® Xeon® Processor Scalable Family is based on the Cascade Lake product and supports Intel 64 architecture.

Some 10th generation Intel® Core™ processors are based on the Ice Lake microarchitecture, and some are based on the Comet Lake microarchitecture; both support Intel 64 architecture.

Some 11th generation Intel® Core™ processors are based on the Tiger Lake microarchitecture, and some are based on the Rocket Lake microarchitecture; both support Intel 64 architecture.

Some 3rd generation Intel® Xeon® Processor Scalable Family processors are based on the Cooper Lake product, and some are based on the Ice Lake microarchitecture; both support Intel 64 architecture.

The 12th generation Intel® Core™ processors are based on the Alder Lake performance hybrid architecture and support Intel 64 architecture.

IA-32 architecture is the instruction set architecture and programming environment for Intel's 32-bit microprocessors. Intel® 64 architecture is the instruction set architecture and programming environment which is the superset of Intel’s 32-bit and 64-bit architectures. It is compatible with the IA-32 architecture.

1.2 OVERVIEW OF THE SYSTEM PROGRAMMING GUIDE

A description of this manual’s content follows:

Chapter 1 — About This Manual. Gives an overview of all eight volumes of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual*. It also describes the notational conventions in these manuals and lists related Intel manuals and documentation of interest to programmers and hardware designers.

Chapter 2 — Model-Specific Registers (MSRs). Lists the MSRs available in Intel processors, and describes their functions.

1.3 NOTATIONAL CONVENTIONS

This manual uses specific notation for data-structure formats, for symbolic representation of instructions, and for hexadecimal and binary numbers. A review of this notation makes the manual easier to read.

1.3.1 Bit and Byte Order

In illustrations of data structures in memory, smaller addresses appear toward the bottom of the figure; addresses increase toward the top. Bit positions are numbered from right to left. The numerical value of a set bit is equal to

two raised to the power of the bit position. Intel 64 and IA-32 processors are "little endian" machines; this means the bytes of a word are numbered starting from the least significant byte. Figure 1-1 illustrates these conventions.

1.3.2 Reserved Bits and Software Compatibility

In many register and memory layout descriptions, certain bits are marked as **reserved**. When bits are marked as reserved, it is essential for compatibility with future processors that software treat these bits as having a future, though unknown, effect. The behavior of reserved bits should be regarded as not only undefined, but unpredictable. Software should follow these guidelines in dealing with reserved bits:

- Do not depend on the states of any reserved bits when testing the values of registers which contain such bits. Mask out the reserved bits before testing.
- Do not depend on the states of any reserved bits when storing to memory or to a register.
- Do not depend on the ability to retain information written into any reserved bits.
- When loading a register, always load the reserved bits with the values indicated in the documentation, if any, or reload them with values previously read from the same register.

NOTE

Avoid any software dependence upon the state of reserved bits in Intel 64 and IA-32 registers. Depending upon the values of reserved register bits will make software dependent upon the unspecified manner in which the processor handles these bits. Programs that depend upon reserved values risk incompatibility with future processors.

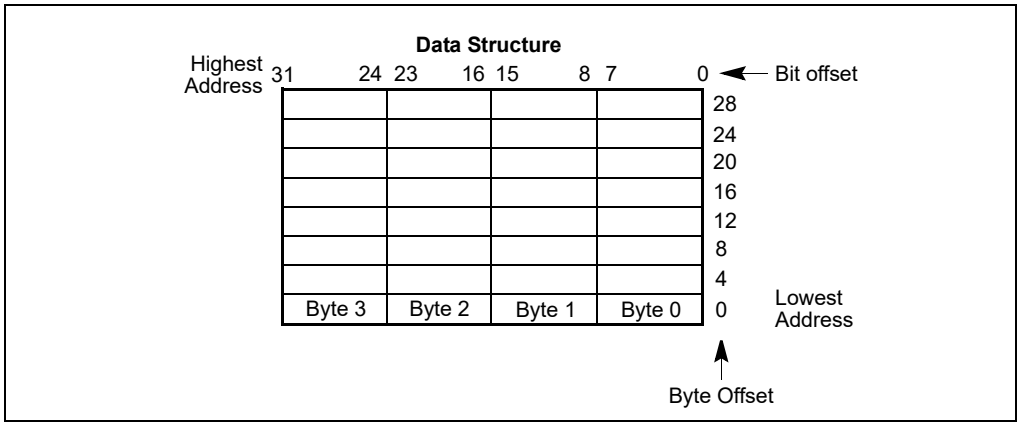


Figure 1-1. Bit and Byte Order

1.3.3 Instruction Operands

When instructions are represented symbolically, a subset of assembly language is used. In this subset, an instruction has the following format:

```
label: mnemonic argument1, argument2, argument3
```

where:

- A **label** is an identifier which is followed by a colon.
- A **mnemonic** is a reserved name for a class of instruction opcodes which have the same function.
- The operands **argument1**, **argument2**, and **argument3** are optional. There may be from zero to three operands, depending on the opcode. When present, they take the form of either literals or identifiers for data items. Operand identifiers are either reserved names of registers or are assumed to be assigned to data items declared in another part of the program (which may not be shown in the example).

When two operands are present in an arithmetic or logical instruction, the right operand is the source and the left operand is the destination.

For example:

```
LOADREG: MOV EAX, SUBTOTAL
```

In this example LOADREG is a label, MOV is the mnemonic identifier of an opcode, EAX is the destination operand, and SUBTOTAL is the source operand. Some assembly languages put the source and destination in reverse order.

1.3.4 Hexadecimal and Binary Numbers

Base 16 (hexadecimal) numbers are represented by a string of hexadecimal digits followed by the character H (for example, F82EH). A hexadecimal digit is a character from the following set: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Base 2 (binary) numbers are represented by a string of 1s and 0s, sometimes followed by the character B (for example, 1010B). The "B" designation is only used in situations where confusion as to the type of number might arise.

1.3.5 Segmented Addressing

The processor uses byte addressing. This means memory is organized and accessed as a sequence of bytes. Whether one or more bytes are being accessed, a byte address is used to locate the byte or bytes memory. The range of memory that can be addressed is called an **address space**.

The processor also supports segmented addressing. This is a form of addressing where a program may have many independent address spaces, called **segments**. For example, a program can keep its code (instructions) and stack in separate segments. Code addresses would always refer to the code space, and stack addresses would always refer to the stack space. The following notation is used to specify a byte address within a segment:

Segment-register:Byte-address

For example, the following segment address identifies the byte at address FF79H in the segment pointed by the DS register:

```
DS:FF79H
```

The following segment address identifies an instruction address in the code segment. The CS register points to the code segment and the EIP register contains the address of the instruction.

```
CS:EIP
```

1.3.6 Syntax for CPUID, CR, and MSR Values

Obtain feature flags, status, and system information by using the CPUID instruction, by checking control register bits, and by reading model-specific registers. We are moving toward a single syntax to represent this type of information. See Figure 1-2.

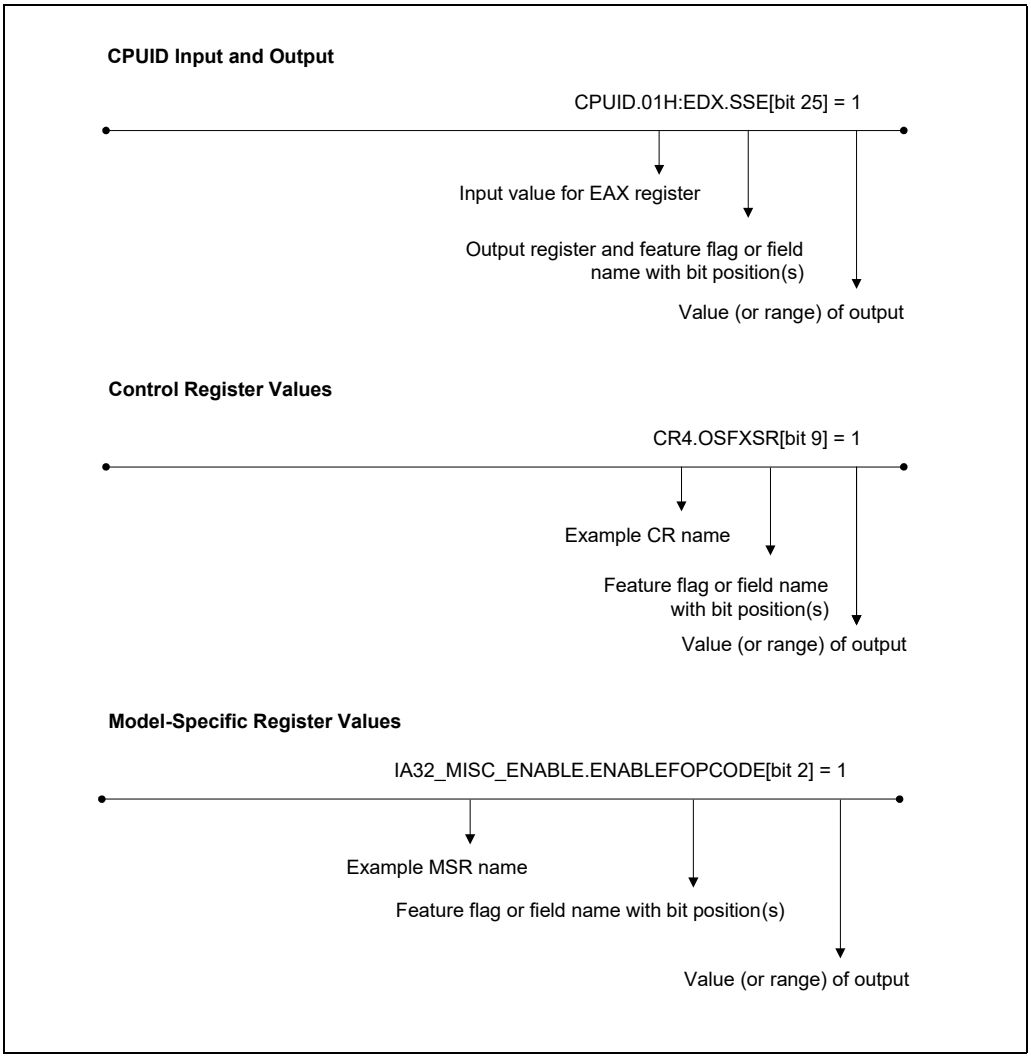


Figure 1-2. Syntax for CPUID, CR, and MSR Data Presentation

1.3.7 Exceptions

An exception is an event that typically occurs when an instruction causes an error. For example, an attempt to divide by zero generates an exception. However, some exceptions, such as breakpoints, occur under other conditions. Some types of exceptions may provide error codes. An error code reports additional information about the error. An example of the notation used to show an exception and error code is shown below:

#PF(fault code)

This example refers to a page-fault exception under conditions where an error code naming a type of fault is reported. Under some conditions, exceptions which produce error codes may not be able to report an accurate code. In this case, the error code is zero, as shown below for a general-protection exception:

#GP(0)

1.4 RELATED LITERATURE

Literature related to Intel 64 and IA-32 processors is listed and viewable on-line at:

https://software.intel.com/en-us/articles/intel-sdm

See also:

- The latest security information on Intel® products:
  https://www.intel.com/content/www/us/en/security-center/default.html
- Software developer resources, guidance and insights for security advisories:
  https://software.intel.com/security-software-guidance/
- The data sheet for a particular Intel 64 or IA-32 processor
- The specification update for a particular Intel 64 or IA-32 processor
- Intel® C++ Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Fortran Compiler documentation and online help:
  http://software.intel.com/en-us/articles/intel-compilers/
- Intel® Software Development Tools:
  https://software.intel.com/en-us/intel-sdp-home
- Intel® 64 and IA-32 Architectures Software Developer's Manual (in one, four or ten volumes):
  https://software.intel.com/en-us/articles/intel-sdm
- Intel® 64 and IA-32 Architectures Optimization Reference Manual:
  https://software.intel.com/en-us/articles/intel-sdm#optimization
- Intel® Trusted Execution Technology Measured Launched Environment Programming Guide:
  http://www.intel.com/content/www/us/en/software-developers/intel-txt-software-development-guide.html
- Intel® Software Guard Extensions (Intel® SGX) Information
  https://software.intel.com/en-us/isa-extensions/intel-sgx
- Developing Multi-threaded Applications: A Platform Consistent Approach:
  https://software.intel.com/sites/default/files/article/147714/51534-developing-multithreaded-applications.pdf
- Using Spin-Loops on Intel® Pentium® 4 Processor and Intel® Xeon® Processor:
  https://software.intel.com/sites/default/files/22/30/25602
- Performance Monitoring Unit Sharing Guide
  http://software.intel.com/file/30388

Literature related to select features in future Intel processors are available at:

- Intel® Architecture Instruction Set Extensions Programming Reference
  https://software.intel.com/en-us/isa-extensions

More relevant links are:

- Intel® Developer Zone:
  https://software.intel.com/en-us
- Developer centers:
  http://www.intel.com/content/www/us/en/hardware-developers/developer-centers.html
- Processor support general link:
  http://www.intel.com/support/processors/
- Intel® Hyper-Threading Technology (Intel® HT Technology):
  http://www.intel.com/technology/platform-technology/hyper-threading/index.htm

29. Updates to Chapter 2, Volume 4

Change bars show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4:* Model-Specific Registers (MSRs).

Changes to this chapter: CPUID family/model value added for 12th generation Intel Core processor. CPUID family/model value added for Rocket Lake, Elkhart Lake and Jasper Lake products. CPUID family/model value added for future Intel Xeon processors. IA32_PPIN_CTL and IA32_PPIN MSRs added to Table 2-2 (Architectural MSRs). MSR names in other tables updated to use "IA32" prefix since MSRs moved to architectural. Updated IA32_ARCH_CAPABILITIES MSR (10AH) in Table 2-2 to add bits 10 and 11. Updated instances of MSR_PEBS_ENABLE to IA32_PEBS_ENABLE and kept former name in parenthesis. Updated the encryption algorithms in the following MSRs : IA32_TME_CAPABILITY and IA32_TME_ACTIVATE. Added MSRs 492H and 493H (IA32_VMX_PROCBASED_CTLS3 and IA32_VMX_EXIT_CTLS2) and updated MSR 483H (IA32_VMX_EXIT_CTLS) in Table 2-2. Updated comment field in architectural MSRs (name line and bit 55 line): IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_STATUS_RESET, IA32_PERF_GLOBAL_STATUS_SET. Corrected typos in Tremont MSR list. Added Alder Lake MSRs and Intel In-field Scan MSRs.Various updates and corrections throughout chapter and MSR lists.

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 Leaf 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 8.9.1, "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 leaf function 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_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_8FH	Future Intel® Xeon® processors based on Sapphire Rapids microarchitecture
06_97H, 06_9AH, 06_BFH	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, 06_A8H	11th generation Intel® Core™ processors based on Rocket Lake microarchitecture
06_7DH, 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® Processor Scalable Family based on Ice Lake microarchitecture
06_55H	Intel® Xeon® Processor Scalable Family based on Skylake microarchitecture, 2nd generation Intel® Xeon® Processor Scalable Family based on Cascade Lake product, and 3rd generation Intel® Xeon® Processor Scalable 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_56H	Intel Xeon processor D-1500 product family based on Broadwell microarchitecture

Table 2-1. CPUID Signature (Contd.)Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
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
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

Table 2-1. CPUID Signature (Contd.)Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number 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

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 reported by CPUID.8000_0008H leaf.

MSR address range between 40000000H - 400000FFH 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		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
0H	0	IA32_P5_MC_ADDR (P5_MC_ADDR)	See Section 2.23, "MSRs in Pentium Processors."	Pentium Processor (05_01H)
1H	1	IA32_P5_MC_TYPE (P5_MC_TYPE)	See Section 2.23, "MSRs in Pentium Processors."	DF_DM = 05_01H
6H	6	IA32_MONITOR_FILTER_SIZE	See Section 8.10.5, "Monitor/Mwait Address Range Determination."	0F_03H

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
10H	16	IA32_TIME_STAMP_COUNTER (TSC)	See Section 17.17, "Time-Stamp Counter."	05_01H
17H	23	IA32_PLATFORM_ID (MSR_PLATFORM_ID)	Platform ID (RO) 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 (RO) 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	
1BH	27	IA32_APIC_BASE (APIC_BASE)	This register holds the APIC base address, permitting the relocation of the APIC memory map. See Section 10.4.4, "Local APIC Status and Location" and Section 10.4.5, "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)	
		(MAXPHYADDR - 1):12	APIC Base (R/W)	
		63: MAXPHYADDR	Reserved	
3AH	58	IA32_FEATURE_CONTROL	Control Features in Intel 64 Processor (R/W)	If any one enumeration condition for defined bit field holds.

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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 the CPUID function 1 returns VMX feature flag and SMX feature flag set (ECX bits 5 and 6 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 the CPUID function 1 returns the VMX feature flag set (ECX bit 5).	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.1:ECX.[bit 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.1:ECX.[bit 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.(EAX=07H, ECX=0H): ECX[30] = 1
		18	SGX Global Enable (R/WL): This bit must be set to enable SGX leaf functions.	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1
		19	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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	
3BH	59	IA32_TSC_ADJUST	Per Logical Processor TSC Adjust (R/Write to clear)	If CPUID.(EAX=07H, ECX=0H): 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.	
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.(EAX=07H, ECX=0):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.(EAX=07H, ECX=0):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.(EAX=07H, ECX=0):EDX[31]=1
		63:3	Reserved	
49H	73	IA32_PRED_CMD	Prediction Command (WO) 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.(EAX=07H, ECX=0):EDX[26]=1
		63:1	Reserved	
4EH	78	IA32_PPIN_CTL	Protected Processor Inventory Number Enable Control (R/W)	If CPUID.(EAX=07H, ECX=01H):EBX[0]=1[1]

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	LockOut (R/WO) 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	
4FH	79	IA32_PPIN	Protected Processor Inventory Number (R/O)	If CPUID.(EAX=07H, ECX=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'.	
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 9.11.6, "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
8BH	139	IA32_BIOS_SIGN_ID (BIOS_SIGN/BBL_CR_D3)	BIOS Update Signature (RO) 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	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:32	It is recommended that this field be pre-loaded 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.	
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.(EAX=12H,ECX=0H): EAX[0]=1 && CPUID.(EAX=07H, ECX=0H):ECX[30]=1. Write permitted if CPUID.(EAX=12H,ECX=0H): EAX[0]=1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1.
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.	
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.	
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.	
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 31.14.4).	If IA32_VMX_MISC[28]
		11:3	Reserved	
		31:12	MSEG Base (R/W)	
		63:32	Reserved	
9EH	158	IA32_SMBASE	Base address of the logical processor's SMRAM image (RO, SMM only).	If IA32_VMX_MISC[15]
BCH	188	IA32_MISC_PACKAGE_CTLS	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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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.	
C1H	193	IA32_PMC0 (PERFCTR0)	General Performance Counter 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0
C2H	194	IA32_PMC1 (PERFCTR1)	General Performance Counter 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1
C3H	195	IA32_PMC2	General Performance Counter 2 (R/W)	If CPUID.0AH: EAX[15:8] > 2
C4H	196	IA32_PMC3	General Performance Counter 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
C5H	197	IA32_PMC4	General Performance Counter 4 (R/W)	If CPUID.0AH: EAX[15:8] > 4
C6H	198	IA32_PMC5	General Performance Counter 5 (R/W)	If CPUID.0AH: EAX[15:8] > 5
C7H	199	IA32_PMC6	General Performance Counter 6 (R/W)	If CPUID.0AH: EAX[15:8] > 6
C8H	200	IA32_PMC7	General Performance Counter 7 (R/W)	If CPUID.0AH: EAX[15:8] > 7
CFH	207	IA32_CORE_CAPABILITIES	IA32 Core Capabilities Register	If CPUID.(EAX=07H, ECX=0):EDX[30] = 1
		63:0	Reserved.	No architecturally defined bits.
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.	
E7H	231	IA32_MPERF	TSC Frequency Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:0	C0_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.	
E8H	232	IA32_APERF	Actual Performance Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1
		63:0	C0_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.	
FEH	254	IA32_MTRRCAP (MTRRcap)	MTRR Capability (RO) See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."	06_01H
		7:0	VCNT: The number of variable memory type ranges in the processor.	
		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.	
		63:13	Reserved	
10AH	266	IA32_ARCH_CAPABILITIES	Enumeration of Architectural Features (RO)	If CPUID.(EAX=07H, ECX=0):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).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		6	IF_PSCHANGE_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	Reserved	
		10	MISC_PACKAGE_CTLS: The processor supports IA32_MISC_PACKAGE_CTLS MSR.	
		11	ENERGY_FILTERING_CTL: The processor supports setting and reading the IA32_MISC_PACKAGE_CTLS[0] (ENERGY_FILTERING_ENABLE) bit.	
		63:12	Reserved	
10BH	267	IA32_FLUSH_CMD	Flush Command (WO) 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.(EAX=07H, ECX=0):EDX[28]=1
		63:1	Reserved	
122H	290	IA32_TSX_CTRL	IA32_TSX_CTRL	Thread scope. Not architecturally serializing. Available when CPUID.ARCH_CAP(EAX=7H, ECX = 0):EDX[29] = 1 and IA32_ARCH_CAPABILITIES. bit 7 = 1.
		0	RTM_DISABLE: When set to 1, XBEGIN will always abort with EAX code 0.	
		1	TSX_CPUID_CLEAR: When set to 1, CPUID.07H.EBX.RTM [bit 11] and CPUID.07H.EBX.HLE [bit 4] report 0. When set to 0 and the SKU supports TSX, these bits will return 1.	
		63:2	Reserved	
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.
175H	373	IA32_SYSENTER_ESP	SYSENTER_ESP_MSR (R/W)	06_01H

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
176H	374	IA32_SYSENTER_EIP	SYSENTER_EIP_MSR (R/W)	06_01H
179H	377	IA32_MCG_CAP (MCG_CAP)	Global Machine Check Capability (RO)	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.	
		15:12	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
		63:28	Reserved	
17AH	378	IA32_MCG_STATUS (MCG_STATUS)	Global Machine Check Status (R/W0)	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	If IA32_MCG_CAP.LMCE_P[27] =1
		63:4	Reserved	
17BH	379	IA32_MCG_CTL (MCG_CTL)	Global Machine Check Control (R/W)	If IA32_MCG_CAP.CTL_P[8] =1
180H-185H	384-389	Reserved		06_0EH[2]

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
186H	390	IA32_PERFEVTSEL0 (PERFEVTSEL0)	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	
187H	391	IA32_PERFEVTSEL1 (PERFEVTSEL1)	Performance Event Select Register 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1
188H	392	IA32_PERFEVTSEL2	Performance Event Select Register 2 (R/W)	If CPUID.0AH: EAX[15:8] > 2
189H	393	IA32_PERFEVTSEL3	Performance Event Select Register 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
18AH	394	IA32_PERFEVTSEL4	Performance Event Select Register 4 (R/W)	If CPUID.0AH: EAX[15:8] > 4
18BH	395	IA32_PERFEVTSEL5	Performance Event Select Register 5 (R/W)	If CPUID.0AH: EAX[15:8] > 5
18CH	396	IA32_PERFEVTSEL6	Performance Event Select Register 6 (R/W)	If CPUID.0AH: EAX[15:8] > 6
18DH	397	IA32_PERFEVTSEL7	Performance Event Select Register 7 (R/W)	If CPUID.0AH: EAX[15:8] > 7

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
18AH-197H	394-407	Reserved		06_0EH[3]
198H	408	IA32_PERF_STATUS	Current Performance Status (RO) See Section 14.1.1, "Software Interface For Initiating Performance State Transitions".	0F_03H
		15:0	Current performance State Value.	
		63:16	Reserved	
199H	409	IA32_PERF_CTL	Performance Control MSR (R/W) Software makes a request for a new Performance state (P-State) by writing this MSR. See Section 14.1.1, "Software Interface For Initiating Performance State Transitions".	0F_03H
		15:0	Target performance State Value.	
		31:16	Reserved	
		32	IDA Engage (R/W) When set to 1: disengages IDA.	06_0FH (Mobile only)
		63:33	Reserved	
19AH	410	IA32_CLOCK_MODULATION	Clock Modulation Control (R/W) See Section 14.8.3, "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	
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 14.8.2, "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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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	
19CH	412	IA32_THERM_STATUS	Thermal Status Information (RO) Contains status information about the processor's thermal sensor and automatic thermal monitoring facilities. See Section 14.8.2, "Thermal Monitor".	If CPUID.01H:EDX[22] = 1
		0	Thermal Status (RO)	If CPUID.01H:EDX[22] = 1
		1	Thermal Status Log (R/W)	If CPUID.01H:EDX[22] = 1
		2	PROCHOT # or FORCEPR# event (RO)	If CPUID.01H:EDX[22] = 1
		3	PROCHOT # or FORCEPR# log (R/WC0)	If CPUID.01H:EDX[22] = 1
		4	Critical Temperature Status (RO)	If CPUID.01H:EDX[22] = 1
		5	Critical Temperature Status log (R/WC0)	If CPUID.01H:EDX[22] = 1
		6	Thermal Threshold #1 Status (RO)	If CPUID.01H:ECX[8] = 1
		7	Thermal Threshold #1 log (R/WC0)	If CPUID.01H:ECX[8] = 1
		8	Thermal Threshold #2 Status (RO)	If CPUID.01H:ECX[8] = 1
		9	Thermal Threshold #2 log (R/WC0)	If CPUID.01H:ECX[8] = 1
		10	Power Limitation Status (RO)	If CPUID.06H:EAX[4] = 1
		11	Power Limitation log (R/WC0)	If CPUID.06H:EAX[4] = 1
		12	Current Limit Status (RO)	If CPUID.06H:EAX[7] = 1
		13	Current Limit log (R/WC0)	If CPUID.06H:EAX[7] = 1
		14	Cross Domain Limit Status (RO)	If CPUID.06H:EAX[7] = 1
		15	Cross Domain Limit log (R/WC0)	If CPUID.06H:EAX[7] = 1
		22:16	Digital Readout (RO)	If CPUID.06H:EAX[0] = 1
		26:23	Reserved	
		30:27	Resolution in Degrees Celsius (RO)	If CPUID.06H:EAX[0] = 1
		31	Reading Valid (RO)	If CPUID.06H:EAX[0] = 1
		63:32	Reserved	
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 MOVS and REP STORS) is enabled (default). When clear, fast-strings are disabled.	0F_0H

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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.	0F_0H
		6:4	Reserved	
		7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.	0F_0H
		10:8	Reserved	
		11	Branch Trace Storage Unavailable (RO) 1 = Processor doesn't support branch trace storage (BTS). 0 = BTS is supported.	0F_0H
		12	Processor Event Based Sampling (PEBS) Unavailable (RO) 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	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		18	ENABLE MONITOR FSM (R/W) When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[bit 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[bit 3] = 1). If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[bit 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. BIOS should contain a setup question that allows users to specify when the installed OS does not support CPUID functions greater than 2. Before setting this bit, BIOS must execute the CPUID.0H and examine the maximum value returned in EAX[7:0]. If the maximum value is greater than 2, this bit is supported. Otherwise, this bit is not supported. Setting this bit when the maximum value is not greater than 2 may generate a #GP exception. Setting this bit may cause unexpected behavior in software that depends on the availability of CPUID leaves greater than 2.	0F_03H
		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
		33:24	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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.	If CPUID.80000001H:EDX[20] = 1
		63:35	Reserved	
1B0H	432	IA32_ENERGY_PERF_BIAS	Performance Energy Bias Hint (R/W)	If CPUID.6H:ECX[3] = 1
		3:0	Power Policy Preference: 0 indicates preference to highest performance. 15 indicates preference to maximize energy saving.	
		63:4	Reserved	
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package Thermal Status Information (RO) Contains status information about the package's thermal sensor. See Section 14.9, "Package Level Thermal Management."	If CPUID.06H: EAX[6] = 1
		0	Pkg Thermal Status (RO)	
		1	Pkg Thermal Status Log (R/W)	
		2	Pkg PROCHOT # event (RO)	
		3	Pkg PROCHOT # log (R/WC0)	
		4	Pkg Critical Temperature Status (RO)	
		5	Pkg Critical Temperature Status Log (R/WC0)	
		6	Pkg Thermal Threshold #1 Status (RO)	
		7	Pkg Thermal Threshold #1 Log (R/WC0)	
		8	Pkg Thermal Threshold #2 Status (RO)	
		9	Pkg Thermal Threshold #1 Log (R/WC0)	
		10	Pkg Power Limitation Status (RO)	
		11	Pkg Power Limitation Log (R/WC0)	
		15:12	Reserved	
		22:16	Pkg Digital Readout (RO)	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		25:23	Reserved	
		26	Hardware Feedback Interface Structure Change Status	If CPUID.06H:EAX.[19] = 1
		63:27	Reserved	
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 14.9, "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	
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
		5:2	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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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
		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.(EAX=07H, ECX=0):EBX[11] = 1)
		63:16	Reserved	
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
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
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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel® TSX (CPUID.07H.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 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.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 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
1F2H	498	IA32_SMRR_PHYSBASE	SMRR Base Address (Writeable only in SMM) Base address of SMM memory range.	If IA32_MTRRCAP.SMRR[11] = 1
		7:0	Type. Specifies memory type of the range.	
		11:8	Reserved	
		31:12	PhysBase SMRR physical Base Address.	
		63:32	Reserved	
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	
1F8H	504	IA32_PLATFORM_DCA_CAP	DCA Capability (R)	If CPUID.01H: ECX[18] = 1
1F9H	505	IA32_CPU_DCA_CAP	If set, CPU supports Prefetch-Hint type.	If CPUID.01H: ECX[18] = 1
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	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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. CR0.CD = 1).	
		31:27	Reserved	
200H	512	IA32_MTRR_PHYSBASE0 (MTRRphysBase0)	See Section 11.11.2.3, "Variable Range MTRRs."	If IA32_MTRRCAP[7:0] > 0
201H	513	IA32_MTRR_PHYSMASK0	MTRRphysMask0	If IA32_MTRRCAP[7:0] > 0
202H	514	IA32_MTRR_PHYSBASE1	MTRRphysBase1	If IA32_MTRRCAP[7:0] > 1
203H	515	IA32_MTRR_PHYSMASK1	MTRRphysMask1	If IA32_MTRRCAP[7:0] > 1
204H	516	IA32_MTRR_PHYSBASE2	MTRRphysBase2	If IA32_MTRRCAP[7:0] > 2
205H	517	IA32_MTRR_PHYSMASK2	MTRRphysMask2	If IA32_MTRRCAP[7:0] > 2
206H	518	IA32_MTRR_PHYSBASE3	MTRRphysBase3	If IA32_MTRRCAP[7:0] > 3
207H	519	IA32_MTRR_PHYSMASK3	MTRRphysMask3	If IA32_MTRRCAP[7:0] > 3
208H	520	IA32_MTRR_PHYSBASE4	MTRRphysBase4	If IA32_MTRRCAP[7:0] > 4
209H	521	IA32_MTRR_PHYSMASK4	MTRRphysMask4	If IA32_MTRRCAP[7:0] > 4
20AH	522	IA32_MTRR_PHYSBASE5	MTRRphysBase5	If IA32_MTRRCAP[7:0] > 5
20BH	523	IA32_MTRR_PHYSMASK5	MTRRphysMask5	If IA32_MTRRCAP[7:0] > 5
20CH	524	IA32_MTRR_PHYSBASE6	MTRRphysBase6	If IA32_MTRRCAP[7:0] > 6
20DH	525	IA32_MTRR_PHYSMASK6	MTRRphysMask6	If IA32_MTRRCAP[7:0] > 6
20EH	526	IA32_MTRR_PHYSBASE7	MTRRphysBase7	If IA32_MTRRCAP[7:0] > 7
20FH	527	IA32_MTRR_PHYSMASK7	MTRRphysMask7	If IA32_MTRRCAP[7:0] > 7
210H	528	IA32_MTRR_PHYSBASE8	MTRRphysBase8	If IA32_MTRRCAP[7:0] > 8
211H	529	IA32_MTRR_PHYSMASK8	MTRRphysMask8	If IA32_MTRRCAP[7:0] > 8
212H	530	IA32_MTRR_PHYSBASE9	MTRRphysBase9	If IA32_MTRRCAP[7:0] > 9
213H	531	IA32_MTRR_PHYSMASK9	MTRRphysMask9	If IA32_MTRRCAP[7:0] > 9
250H	592	IA32_MTRR_FIX64K_00000	MTRRfix64K_00000	If CPUID.01H: EDX.MTRR[12] =1
258H	600	IA32_MTRR_FIX16K_80000	MTRRfix16K_80000	If CPUID.01H: EDX.MTRR[12] =1
259H	601	IA32_MTRR_FIX16K_A0000	MTRRfix16K_A0000	If CPUID.01H: EDX.MTRR[12] =1
268H	616	IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000)	See Section 11.11.2.2, "Fixed Range MTRRs."	If CPUID.01H: EDX.MTRR[12] =1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
269H	617	IA32_MTRR_FIX4K_C8000	MTRRfix4K_C8000	If CPUID.01H: EDX.MTRR[12] =1
26AH	618	IA32_MTRR_FIX4K_D0000	MTRRfix4K_D0000	If CPUID.01H: EDX.MTRR[12] =1
26BH	619	IA32_MTRR_FIX4K_D8000	MTRRfix4K_D8000	If CPUID.01H: EDX.MTRR[12] =1
26CH	620	IA32_MTRR_FIX4K_E0000	MTRRfix4K_E0000	If CPUID.01H: EDX.MTRR[12] =1
26DH	621	IA32_MTRR_FIX4K_E8000	MTRRfix4K_E8000	If CPUID.01H: EDX.MTRR[12] =1
26EH	622	IA32_MTRR_FIX4K_F0000	MTRRfix4K_F0000	If CPUID.01H: EDX.MTRR[12] =1
26FH	623	IA32_MTRR_FIX4K_F8000	MTRRfix4K_F8000	If CPUID.01H: EDX.MTRR[12] =1
277H	631	IA32_PAT	IA32_PAT (R/W)	If CPUID.01H: EDX.MTRR[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	
280H	640	IA32_MC0_CTL2	MSR to enable/disable CMCI capability for bank 0. (R/W) See Section 15.3.2.5, "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	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
281H	641	IA32_MC1_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 1
282H	642	IA32_MC2_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 2
283H	643	IA32_MC3_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 3
284H	644	IA32_MC4_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 4
285H	645	IA32_MC5_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 5
286H	646	IA32_MC6_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 6
287H	647	IA32_MC7_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 7
288H	648	IA32_MC8_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 8
289H	649	IA32_MC9_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 9
28AH	650	IA32_MC10_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 10
28BH	651	IA32_MC11_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 11
28CH	652	IA32_MC12_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 12
28DH	653	IA32_MC13_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 13
28EH	654	IA32_MC14_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 14
28FH	655	IA32_MC15_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 15

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
290H	656	IA32_MC16_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 16
291H	657	IA32_MC17_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 17
292H	658	IA32_MC18_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 18
293H	659	IA32_MC19_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 19
294H	660	IA32_MC20_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 20
295H	661	IA32_MC21_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 21
296H	662	IA32_MC22_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 22
297H	663	IA32_MC23_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 23
298H	664	IA32_MC24_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 24
299H	665	IA32_MC25_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 25
29AH	666	IA32_MC26_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 26
29BH	667	IA32_MC27_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 27
29CH	668	IA32_MC28_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 28
29DH	669	IA32_MC29_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 29
29EH	670	IA32_MC30_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 30

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
29FH	671	IA32_MC31_CTL2	(R/W) Same fields as IA32_MC0_CTL2.	If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 31
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	
309H	777	IA32_FIXED_CTR0	Fixed-Function Performance Counter 0 (R/W): Counts Instr_Retired.Any.	If CPUID.0AH: EDX[4:0] > 0
30AH	778	IA32_FIXED_CTR1	Fixed-Function Performance Counter 1 (R/W): Counts CPU_CLK_Unhalted.Core.	If CPUID.0AH: EDX[4:0] > 1
30BH	779	IA32_FIXED_CTR2	Fixed-Function Performance Counter 2 (R/W): Counts CPU_CLK_Unhalted.Ref.	If CPUID.0AH: EDX[4:0] > 2
345H	837	IA32_PERF_CAPABILITIES	Read Only MSR that enumerates the existence of performance monitoring features. (RO)	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.0x7.0.EBX[25]=1
		63:17	Reserved	
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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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 1to count while CPL = 0.	
		5	EN1_Usr: Enable Fixed Counter 1to 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	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
38EH	910	IA32_PERF_GLOBAL_STATUS	Global Performance Counter Status (RO)	If CPUID.0AH: EAX[7:0] > 0 II (CPUID.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):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
		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
		47:35	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.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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 CPUID.(EAX=07H, ECX=0):EBX[2] = 1
		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
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
		47:35	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	
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Global Performance Counter Overflow Control (R/W)	If CPUID.0AH: EAX[7:0] > 0 && CPUID.0AH: EAX[7:0] <= 3
		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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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
		54:35	Reserved	
		55	Set 1 to Clear Trace_ToPA_PMI bit.	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && IA32_RTIT_CTL.ToPA = 1
		60:56	Reserved	
		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
390H	912	IA32_PERF_GLOBAL_STATUS_RESET	Global Performance Counter Overflow Reset Control (R/W)	If CPUID.0AH: EAX[7:0] > 3 II (CPUID.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):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
		47:35	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.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		58	Set 1 to Clear ASCI bit.	If CPUID.0AH: EAX[7:0] > 3
		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
391H	913	IA32_PERF_GLOBAL_STATUS_SET	Global Performance Counter Overflow Set Control (R/W)	If CPUID.0AH: EAX[7:0] > 3 II (CPUID.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):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
		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
		47:35	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.(EAX=07H, ECX=0):EBX[25] = 1 && CPUID.(EAX=014H, ECX=0):ECX[0] = 1
		57:56	Reserved	
		58	Set 1 to cause LBR_Frz = 1.	If CPUID.0AH: EAX[7:0] > 3
		59	Set 1 to cause CTR_Frz = 1.	If CPUID.0AH: EAX[7:0] > 3
		58	Set 1 to cause ASCI = 1.	If CPUID.0AH: EAX[7:0] > 3
		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	
392H	914	IA32_PERF_GLOBAL_INUSE	Indicator that core perfmon interface is in use. (RO)	If CPUID.0AH: EAX[7:0] > 3
		0	IA32_PERFEVTSEL0 in use.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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.	
		62:35	Reserved or model specific.	
		63	PMI in use.	
3F1H	1009	IA32_PEBS_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.	
		63:36	Reserved	
400H	1024	IA32_MC0_CTL	MC0_CTL	If IA32_MCG_CAP.CNT >0
401H	1025	IA32_MC0_STATUS	MC0_STATUS	If IA32_MCG_CAP.CNT >0
402H	1026	IA32_MC0_ADDR[1]	MC0_ADDR	If IA32_MCG_CAP.CNT >0
403H	1027	IA32_MC0_MISC	MC0_MISC	If IA32_MCG_CAP.CNT >0
404H	1028	IA32_MC1_CTL	MC1_CTL	If IA32_MCG_CAP.CNT >1
405H	1029	IA32_MC1_STATUS	MC1_STATUS	If IA32_MCG_CAP.CNT >1
406H	1030	IA32_MC1_ADDR[2]	MC1_ADDR	If IA32_MCG_CAP.CNT >1
407H	1031	IA32_MC1_MISC	MC1_MISC	If IA32_MCG_CAP.CNT >1
408H	1032	IA32_MC2_CTL	MC2_CTL	If IA32_MCG_CAP.CNT >2
409H	1033	IA32_MC2_STATUS	MC2_STATUS	If IA32_MCG_CAP.CNT >2
40AH	1034	IA32_MC2_ADDR[1]	MC2_ADDR	If IA32_MCG_CAP.CNT >2
40BH	1035	IA32_MC2_MISC	MC2_MISC	If IA32_MCG_CAP.CNT >2
40CH	1036	IA32_MC3_CTL	MC3_CTL	If IA32_MCG_CAP.CNT >3
40DH	1037	IA32_MC3_STATUS	MC3_STATUS	If IA32_MCG_CAP.CNT >3
40EH	1038	IA32_MC3_ADDR[1]	MC3_ADDR	If IA32_MCG_CAP.CNT >3
40FH	1039	IA32_MC3_MISC	MC3_MISC	If IA32_MCG_CAP.CNT >3
410H	1040	IA32_MC4_CTL	MC4_CTL	If IA32_MCG_CAP.CNT >4
411H	1041	IA32_MC4_STATUS	MC4_STATUS	If IA32_MCG_CAP.CNT >4
412H	1042	IA32_MC4_ADDR[1]	MC4_ADDR	If IA32_MCG_CAP.CNT >4
413H	1043	IA32_MC4_MISC	MC4_MISC	If IA32_MCG_CAP.CNT >4

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
414H	1044	IA32_MC5_CTL	MC5_CTL	If IA32_MCG_CAP.CNT >5
415H	1045	IA32_MC5_STATUS	MC5_STATUS	If IA32_MCG_CAP.CNT >5
416H	1046	IA32_MC5_ADDR[1]	MC5_ADDR	If IA32_MCG_CAP.CNT >5
417H	1047	IA32_MC5_MISC	MC5_MISC	If IA32_MCG_CAP.CNT >5
418H	1048	IA32_MC6_CTL	MC6_CTL	If IA32_MCG_CAP.CNT >6
419H	1049	IA32_MC6_STATUS	MC6_STATUS	If IA32_MCG_CAP.CNT >6
41AH	1050	IA32_MC6_ADDR[1]	MC6_ADDR	If IA32_MCG_CAP.CNT >6
41BH	1051	IA32_MC6_MISC	MC6_MISC	If IA32_MCG_CAP.CNT >6
41CH	1052	IA32_MC7_CTL	MC7_CTL	If IA32_MCG_CAP.CNT >7
41DH	1053	IA32_MC7_STATUS	MC7_STATUS	If IA32_MCG_CAP.CNT >7
41EH	1054	IA32_MC7_ADDR[1]	MC7_ADDR	If IA32_MCG_CAP.CNT >7
41FH	1055	IA32_MC7_MISC	MC7_MISC	If IA32_MCG_CAP.CNT >7
420H	1056	IA32_MC8_CTL	MC8_CTL	If IA32_MCG_CAP.CNT >8
421H	1057	IA32_MC8_STATUS	MC8_STATUS	If IA32_MCG_CAP.CNT >8
422H	1058	IA32_MC8_ADDR[1]	MC8_ADDR	If IA32_MCG_CAP.CNT >8
423H	1059	IA32_MC8_MISC	MC8_MISC	If IA32_MCG_CAP.CNT >8
424H	1060	IA32_MC9_CTL	MC9_CTL	If IA32_MCG_CAP.CNT >9
425H	1061	IA32_MC9_STATUS	MC9_STATUS	If IA32_MCG_CAP.CNT >9
426H	1062	IA32_MC9_ADDR[1]	MC9_ADDR	If IA32_MCG_CAP.CNT >9
427H	1063	IA32_MC9_MISC	MC9_MISC	If IA32_MCG_CAP.CNT >9
428H	1064	IA32_MC10_CTL	MC10_CTL	If IA32_MCG_CAP.CNT >10
429H	1065	IA32_MC10_STATUS	MC10_STATUS	If IA32_MCG_CAP.CNT >10
42AH	1066	IA32_MC10_ADDR[1]	MC10_ADDR	If IA32_MCG_CAP.CNT >10
42BH	1067	IA32_MC10_MISC	MC10_MISC	If IA32_MCG_CAP.CNT >10
42CH	1068	IA32_MC11_CTL	MC11_CTL	If IA32_MCG_CAP.CNT >11
42DH	1069	IA32_MC11_STATUS	MC11_STATUS	If IA32_MCG_CAP.CNT >11
42EH	1070	IA32_MC11_ADDR[1]	MC11_ADDR	If IA32_MCG_CAP.CNT >11
42FH	1071	IA32_MC11_MISC	MC11_MISC	If IA32_MCG_CAP.CNT >11
430H	1072	IA32_MC12_CTL	MC12_CTL	If IA32_MCG_CAP.CNT >12
431H	1073	IA32_MC12_STATUS	MC12_STATUS	If IA32_MCG_CAP.CNT >12
432H	1074	IA32_MC12_ADDR[1]	MC12_ADDR	If IA32_MCG_CAP.CNT >12
433H	1075	IA32_MC12_MISC	MC12_MISC	If IA32_MCG_CAP.CNT >12
434H	1076	IA32_MC13_CTL	MC13_CTL	If IA32_MCG_CAP.CNT >13
435H	1077	IA32_MC13_STATUS	MC13_STATUS	If IA32_MCG_CAP.CNT >13
436H	1078	IA32_MC13_ADDR[1]	MC13_ADDR	If IA32_MCG_CAP.CNT >13
437H	1079	IA32_MC13_MISC	MC13_MISC	If IA32_MCG_CAP.CNT >13
438H	1080	IA32_MC14_CTL	MC14_CTL	If IA32_MCG_CAP.CNT >14

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
439H	1081	IA32_MC14_STATUS	MC14_STATUS	If IA32_MCG_CAP.CNT >14
43AH	1082	IA32_MC14_ADDR[1]	MC14_ADDR	If IA32_MCG_CAP.CNT >14
43BH	1083	IA32_MC14_MISC	MC14_MISC	If IA32_MCG_CAP.CNT >14
43CH	1084	IA32_MC15_CTL	MC15_CTL	If IA32_MCG_CAP.CNT >15
43DH	1085	IA32_MC15_STATUS	MC15_STATUS	If IA32_MCG_CAP.CNT >15
43EH	1086	IA32_MC15_ADDR[1]	MC15_ADDR	If IA32_MCG_CAP.CNT >15
43FH	1087	IA32_MC15_MISC	MC15_MISC	If IA32_MCG_CAP.CNT >15
440H	1088	IA32_MC16_CTL	MC16_CTL	If IA32_MCG_CAP.CNT >16
441H	1089	IA32_MC16_STATUS	MC16_STATUS	If IA32_MCG_CAP.CNT >16
442H	1090	IA32_MC16_ADDR[1]	MC16_ADDR	If IA32_MCG_CAP.CNT >16
443H	1091	IA32_MC16_MISC	MC16_MISC	If IA32_MCG_CAP.CNT >16
444H	1092	IA32_MC17_CTL	MC17_CTL	If IA32_MCG_CAP.CNT >17
445H	1093	IA32_MC17_STATUS	MC17_STATUS	If IA32_MCG_CAP.CNT >17
446H	1094	IA32_MC17_ADDR[1]	MC17_ADDR	If IA32_MCG_CAP.CNT >17
447H	1095	IA32_MC17_MISC	MC17_MISC	If IA32_MCG_CAP.CNT >17
448H	1096	IA32_MC18_CTL	MC18_CTL	If IA32_MCG_CAP.CNT >18
449H	1097	IA32_MC18_STATUS	MC18_STATUS	If IA32_MCG_CAP.CNT >18
44AH	1098	IA32_MC18_ADDR[1]	MC18_ADDR	If IA32_MCG_CAP.CNT >18
44BH	1099	IA32_MC18_MISC	MC18_MISC	If IA32_MCG_CAP.CNT >18
44CH	1100	IA32_MC19_CTL	MC19_CTL	If IA32_MCG_CAP.CNT >19
44DH	1101	IA32_MC19_STATUS	MC19_STATUS	If IA32_MCG_CAP.CNT >19
44EH	1102	IA32_MC19_ADDR[1]	MC19_ADDR	If IA32_MCG_CAP.CNT >19
44FH	1103	IA32_MC19_MISC	MC19_MISC	If IA32_MCG_CAP.CNT >19
450H	1104	IA32_MC20_CTL	MC20_CTL	If IA32_MCG_CAP.CNT >20
451H	1105	IA32_MC20_STATUS	MC20_STATUS	If IA32_MCG_CAP.CNT >20
452H	1106	IA32_MC20_ADDR[1]	MC20_ADDR	If IA32_MCG_CAP.CNT >20
453H	1107	IA32_MC20_MISC	MC20_MISC	If IA32_MCG_CAP.CNT >20
454H	1108	IA32_MC21_CTL	MC21_CTL	If IA32_MCG_CAP.CNT >21
455H	1109	IA32_MC21_STATUS	MC21_STATUS	If IA32_MCG_CAP.CNT >21
456H	1110	IA32_MC21_ADDR[1]	MC21_ADDR	If IA32_MCG_CAP.CNT >21
457H	1111	IA32_MC21_MISC	MC21_MISC	If IA32_MCG_CAP.CNT >21
458H	1112	IA32_MC22_CTL	MC22_CTL	If IA32_MCG_CAP.CNT >22
459H	1113	IA32_MC22_STATUS	MC22_STATUS	If IA32_MCG_CAP.CNT >22
45AH	1114	IA32_MC22_ADDR[1]	MC22_ADDR	If IA32_MCG_CAP.CNT >22
45BH	1115	IA32_MC22_MISC	MC22_MISC	If IA32_MCG_CAP.CNT >22
45CH	1116	IA32_MC23_CTL	MC23_CTL	If IA32_MCG_CAP.CNT >23
45DH	1117	IA32_MC23_STATUS	MC23_STATUS	If IA32_MCG_CAP.CNT >23

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
45EH	1118	IA32_MC23_ADDR[1]	MC23_ADDR	If IA32_MCG_CAP.CNT >23
45FH	1119	IA32_MC23_MISC	MC23_MISC	If IA32_MCG_CAP.CNT >23
460H	1120	IA32_MC24_CTL	MC24_CTL	If IA32_MCG_CAP.CNT >24
461H	1121	IA32_MC24_STATUS	MC24_STATUS	If IA32_MCG_CAP.CNT >24
462H	1122	IA32_MC24_ADDR[1]	MC24_ADDR	If IA32_MCG_CAP.CNT >24
463H	1123	IA32_MC24_MISC	MC24_MISC	If IA32_MCG_CAP.CNT >24
464H	1124	IA32_MC25_CTL	MC25_CTL	If IA32_MCG_CAP.CNT >25
465H	1125	IA32_MC25_STATUS	MC25_STATUS	If IA32_MCG_CAP.CNT >25
466H	1126	IA32_MC25_ADDR[1]	MC25_ADDR	If IA32_MCG_CAP.CNT >25
467H	1127	IA32_MC25_MISC	MC25_MISC	If IA32_MCG_CAP.CNT >25
468H	1128	IA32_MC26_CTL	MC26_CTL	If IA32_MCG_CAP.CNT >26
469H	1129	IA32_MC26_STATUS	MC26_STATUS	If IA32_MCG_CAP.CNT >26
46AH	1130	IA32_MC26_ADDR[1]	MC26_ADDR	If IA32_MCG_CAP.CNT >26
46BH	1131	IA32_MC26_MISC	MC26_MISC	If IA32_MCG_CAP.CNT >26
46CH	1132	IA32_MC27_CTL	MC27_CTL	If IA32_MCG_CAP.CNT >27
46DH	1133	IA32_MC27_STATUS	MC27_STATUS	If IA32_MCG_CAP.CNT >27
46EH	1134	IA32_MC27_ADDR[1]	MC27_ADDR	If IA32_MCG_CAP.CNT >27
46FH	1135	IA32_MC27_MISC	MC27_MISC	If IA32_MCG_CAP.CNT >27
470H	1136	IA32_MC28_CTL	MC28_CTL	If IA32_MCG_CAP.CNT >28
471H	1137	IA32_MC28_STATUS	MC28_STATUS	If IA32_MCG_CAP.CNT >28
472H	1138	IA32_MC28_ADDR[1]	MC28_ADDR	If IA32_MCG_CAP.CNT >28
473H	1139	IA32_MC28_MISC	MC28_MISC	If IA32_MCG_CAP.CNT >28
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
481H	1153	IA32_VMX_PINBASED_CTLS	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
482H	1154	IA32_VMX_PROCBASED_CTLS	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
483H	1155	IA32_VMX_EXIT_CTLS	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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
484H	1156	IA32_VMX_ENTRY_CTLS	Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls."	If CPUID.01H:ECX.[5] = 1
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
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
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
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
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
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
48BH	1163	IA32_VMX_PROCBASED_CTLS2	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_CTLS[63])
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_CTLS[63] && (IA32_VMX_PROCBASED_CTLS2[33] || IA32_VMX_PROCBASED_CTLS2[37]))
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	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] = 1 && IA32_VMX_BASIC[55])
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	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] = 1 && IA32_VMX_BASIC[55])
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Appendix A.4, "VM-Exit Controls."	If(CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55])

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Appendix A.5, "VM-Entry Controls."	If(CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55])
491H	1169	IA32_VMX_VMFUNC	Capability Reporting Register of VM-Function Controls (R/O)	If(CPUID.01H:ECX.[5] = 1 && IA32_VMX_BASIC[55])
492H	1170	IA32_VMX_PROCBASED_CTLS3	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_CTLS[49])
493H	1171	IA32_VMX_EXIT_CTLS2	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_CTLS[63)
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
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
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
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
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
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
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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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
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 15.3.1.4, "IA32_MCG_EXT_CTL MSR".	If IA32_MCG_CAP.LMCE_P =1
		0	LMCE_EN	
		63:1	Reserved	
500H	1280	IA32_SGX_SVN_STATUS	Status and SVN Threshold of SGX Support for ACM (RO).	If CPUID.(EAX=07H, ECX=0H): EBX[2] = 1
		0	Lock	See Section 38.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		15:1	Reserved	
		23:16	SGX_SVN_SINIT	See Section 38.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		63:24	Reserved	
560H	1376	IA32_RTIT_OUTPUT_BASE	Trace Output Base Register (R/W)	If ((CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && ((CPUID.(EAX=14H,ECX=0):ECX[0] = 1) || (CPUID.(EAX=14H,ECX=0):ECX[2] = 1)))
		6:0	Reserved	
		MAXPHYADDR[4]-1:7	Base physical address.	
		63:MAXPHYADDR	Reserved	
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Trace Output Mask Pointers Register (R/W)	If ((CPUID.(EAX=07H, ECX=0):EBX[25] = 1) && ((CPUID.(EAX=14H,ECX=0):ECX[0] = 1) || (CPUID.(EAX=14H,ECX=0):ECX[2] = 1)))
		6:0	Reserved	
		31:7	MaskOrTableOffset	
		63:32	Output Offset	
570H	1392	IA32_RTIT_CTL	Trace Control Register (R/W)	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1)
		0	TraceEn	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		1	CYCEn	If (CPUID.(EAX=07H, ECX=0):EBX[1] = 1)
		2	OS	
		3	User	
		4	PwrEvtEn	If (CPUID.(EAX=07H, ECX=1):EBX[5] = 1)
		5	FUPonPTW	If (CPUID.(EAX=07H, ECX=1):EBX[4] = 1)
		6	FabricEn	If (CPUID.(EAX=07H, ECX=0):ECX[3] = 1)
		7	CR3 filter	
		8	ToPA	
		9	MTCEn	If (CPUID.(EAX=07H, ECX=0):EBX[3] = 1)
		10	TSCEn	
		11	DisRETC	
		12	PTWEn	If (CPUID.(EAX=07H, ECX=1):EBX[4] = 1)
		13	BranchEn	
		17:14	MTCFreq	If (CPUID.(EAX=07H, ECX=0):EBX[3] = 1)
		18	Reserved, must be zero.	
		22:19	CYCThresh	If (CPUID.(EAX=07H, ECX=0):EBX[1] = 1)
		23	Reserved, must be zero.	
		27:24	PSBFreq	If (CPUID.(EAX=07H, ECX=0):EBX[1] = 1)
		31:28	Reserved, must be zero.	
		35:32	ADDR0_CFG	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 0)
		39:36	ADDR1_CFG	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 1)
		43:40	ADDR2_CFG	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)
		47:44	ADDR3_CFG	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 3)
		55:48	Reserved, must be zero.	
		56	InjectPsbPmiOnEnable	If (CPUID.(EAX=07H, ECX=1):EBX[6] = 1)
		63:57	Reserved, must be zero.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
571H	1393	IA32_RTIT_STATUS	Tracing Status Register (R/W)	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1)
		0	FilterEn (writes ignored)	If (CPUID.(EAX=07H, ECX=0):EBX[2] = 1)
		1	ContexEn (writes ignored)	
		2	TriggerEn (writes ignored)	
		3	Reserved	
		4	Error	
		5	Stopped	
		6	PendPSB	If (CPUID.(EAX=07H, ECX=0):EBX[6] = 1)
		7	PendToPAPMI	If (CPUID.(EAX=07H, ECX=0):EBX[6] = 1)
		31:8	Reserved, must be zero.	
		48:32	PacketByteCnt	If (CPUID.(EAX=07H, ECX=0):EBX[1] > 3)
		63:49	Reserved	
572H	1394	IA32_RTIT_CR3_MATCH	Trace Filter CR3 Match Register (R/W)	If (CPUID.(EAX=07H, ECX=0):EBX[25] = 1)
		4:0	Reserved	
		63:5	CR3[63:5] value to match.	
580H	1408	IA32_RTIT_ADDR0_A	Region 0 Start Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 0)
		47:0	Virtual Address	
		63:48	SignExt_VA	
581H	1409	IA32_RTIT_ADDR0_B	Region 0 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 0)
		47:0	Virtual Address	
		63:48	SignExt_VA	
582H	1410	IA32_RTIT_ADDR1_A	Region 1 Start Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 1)
		47:0	Virtual Address	
		63:48	SignExt_VA	
583H	1411	IA32_RTIT_ADDR1_B	Region 1 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 1)
		47:0	Virtual Address	
		63:48	SignExt_VA	
584H	1412	IA32_RTIT_ADDR2_A	Region 2 Start Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)
		47:0	Virtual Address	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:48	SignExt_VA	
585H	1413	IA32_RTIT_ADDR2_B	Region 2 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 2)
		47:0	Virtual Address	
		63:48	SignExt_VA	
586H	1414	IA32_RTIT_ADDR3_A	Region 3 Start Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 3)
		47:0	Virtual Address	
		63:48	SignExt_VA	
587H	1415	IA32_RTIT_ADDR3_B	Region 3 End Address (R/W)	If (CPUID.(EAX=07H, ECX=1):EAX[2:0] > 3)
		47:0	Virtual Address	
		63:48	SignExt_VA	
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 19.6.3.4, "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.	
6A0H	1696	IA32_U_CET	Configure User Mode CET (R/W)	Bits 1:0 are defined if CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1. Bits 5:2 and bits 63:10 are defined if CPUID.(EAX=07H, ECX=0H):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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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).	
		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 loaded. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	
6A2H	1698	IA32_S_CET	Configure Supervisor Mode CET (R/W)	See IA32_U_CET (6A0H) for reference; similar format.
6A4H	1700	IA32_PL0_SSP	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. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
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. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
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. The linear address written must be aligned to 8 bytes and bits 2:0 must be 0 (hardware requires bits 1:0 to be 0).	If CPUID.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
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.(EAX=07H, ECX=0H):ECX.CET_SS[07] = 1
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		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
6E1H	1761	IA32_PKRS	Specifies the PK permissions associated with each protection domain for supervisor pages (R/W)	If CPUID.(EAX=07H, ECX=0H):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.	
770H	1904	IA32_PM_ENABLE	Enable/disable HWP (R/W)	If CPUID.06H:EAX.[7] = 1
		0	HWP_ENABLE (R/W1-Once) See Section 14.4.2, "Enabling HWP".	If CPUID.06H:EAX.[7] = 1
		63:1	Reserved	
771H	1905	IA32_HWP_CAPABILITIES	HWP Performance Range Enumeration (RO)	If CPUID.06H:EAX.[7] = 1
		7:0	Highest_Performance See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".	If CPUID.06H:EAX.[7] = 1
		15:8	Guaranteed_Performance See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".	If CPUID.06H:EAX.[7] = 1
		23:16	Most_Efficient_Performance See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".	If CPUID.06H:EAX.[7] = 1
		31:24	Lowest_Performance See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".	If CPUID.06H:EAX.[7] = 1
		63:32	Reserved	
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 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[11] = 1
		15:8	Maximum_Performance See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[11] = 1
		23:16	Desired_Performance See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[11] = 1
		31:24	Energy_Performance_Preference See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[11] = 1 && CPUID.06H:EAX.[10] = 1
		41:32	Activity_Window See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[11] = 1 && CPUID.06H:EAX.[9] = 1
		63:42	Reserved	
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		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	EN_Guaranteed_Performance_Change See Section 14.4.6, "HWP Notifications".	If CPUID.06H:EAX.[8] = 1
		1	EN_Excursion_Minimum See Section 14.4.6, "HWP Notifications".	If CPUID.06H:EAX.[8] = 1
		63:2	Reserved	
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 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1
		15:8	Maximum_Performance See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1
		23:16	Desired_Performance See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1
		31:24	Energy_Performance_Preference See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[10] = 1
		41:32	Activity_Window See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[9] = 1
		42	Package_Control See Section 14.4.4, "Managing HWP".	If CPUID.06H:EAX.[7] = 1 && CPUID.06H:EAX.[11] = 1
		63:43	Reserved	
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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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.	
		63	Min PECI Override (MIN_PECI_OVERRIDE): Indicates whether PECI is currently overriding the Minimum Performance input. If set to '1', PECI is overriding the Minimum Performance input. If clear (0), OS has control over Minimum Performance input. Default value is 0.	
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	
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/WC0) See Section 14.4.5, "HWP Feedback".	If CPUID.06H:EAX.[7] = 1
		1	Reserved	
		2	Excursion_To_Minimum (R/WC0) See Section 14.4.5, "HWP Feedback".	If CPUID.06H:EAX.[7] = 1
		63:3	Reserved	
802H	2050	IA32_X2APIC_APICID	x2APIC ID Register (R/O) See x2APIC Specification.	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
803H	2051	IA32_X2APIC_VERSION	x2APIC Version Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
808H	2056	IA32_X2APIC_TPR	x2APIC Task Priority Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80AH	2058	IA32_X2APIC_PPR	x2APIC Processor Priority Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80BH	2059	IA32_X2APIC_EOI	x2APIC EOI Register (W/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80DH	2061	IA32_X2APIC_LDR	x2APIC Logical Destination Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
80FH	2063	IA32_X2APIC_SIVR	x2APIC Spurious Interrupt Vector Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
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
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
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
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
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
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
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
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
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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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
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
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
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
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
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)
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
820H	2080	IA32_X2APIC_IRR0	x2APIC Interrupt Request Register Bits 31:0 (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
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
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
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
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
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
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
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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
828H	2088	IA32_X2APIC_ESR	x2APIC Error Status Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
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
830H	2096	IA32_X2APIC_ICR	x2APIC Interrupt Command Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
832H	2098	IA32_X2APIC_LVT_TIMER	x2APIC LVT Timer Interrupt Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
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
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
835H	2101	IA32_X2APIC_LVT_LINT0	x2APIC LVT LINT0 Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
836H	2102	IA32_X2APIC_LVT_LINT1	x2APIC LVT LINT1 Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
837H	2103	IA32_X2APIC_LVT_ERROR	x2APIC LVT Error Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
838H	2104	IA32_X2APIC_INIT_COUNT	x2APIC Initial Count Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
839H	2105	IA32_X2APIC_CUR_COUNT	x2APIC Current Count Register (R/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
83EH	2110	IA32_X2APIC_DIV_CONF	x2APIC Divide Configuration Register (R/W)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
83FH	2111	IA32_X2APIC_SELF_IPI	x2APIC Self IPI Register (W/O)	If CPUID.01H:ECX.[21] = 1 && IA32_APIC_BASE.[10] = 1
981H	2433	IA32_TME_CAPABILITY	Memory Encryption Capability MSR	If CPUID.07H: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.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		2	Support for AES-XTS 256-bit encryption algorithm.	
		30:3	Reserved.	
		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 MKTME is not supported.	
		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.	
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:ECX.[13] = 1
		0	Lock RO – Will be set upon successful WRMSR (or first SMI); written value ignored.	
		1	Hardware Encryption Enable This bit also enables MKTME; MKTME cannot be enabled without enabling encryption hardware.	
		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).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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 are allowed. For example: 0000 – AES-XTS-128. 0001 – AES-XTS-128 with integrity. 0010 – AES-XTS-256. Other values are invalid.	
		30:8	Reserved.	
		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. 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 MKTME is not enumerated, otherwise: The number of key identifier bits to allocate to MKTME 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 MKTME. Example: To support 255 keys, this field would be set to a value of 8.	
		47:36	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		63:48	MK_TME_CRYPTO_ALGS Reserved if MKTME is not enumerated, otherwise: Bit 48: AES-XTS 128. Bit 49: AES-XTS 128 with integrity. Bit 50: AES-XTS 256. Bit 63:51: Reserved (#GP) Bitmask for BIOS to set which encryption algorithms are allowed for MKTME, would be later enforced by the key loading ISA ('1 = allowed).	
983H	2435	IA32_TME_EXCLUDE_MASK	Memory Encryption Exclude Mask	If CPUID.07H:ECX.[13] = 1
		10:0	Reserved.	
		11	Enable: When set to '1', then TME_EXCLUDE_BASE and TME_EXCLUDE_MASK are used to define an exclusion region for TME/MKTME (for KeyID=0).	
		MAXPHYSADDR-1:12	TMEEMASK: This field indicates the bits that must match TMEEBASE in order to qualify as a TME/MKTME (for KeyID=0) exclusion memory range access.	
		63:MAXPHYSADDR	Reserved; must be zero.	
984H	2436	IA32_TME_EXCLUDE_BASE	Memory Encryption Exclude Base	If CPUID.07H:ECX.[13] = 1
		11:0	Reserved.	
		MAXPHYSADDR-1:12	TMEEBASE: Base physical address to be excluded for TME/MKTME (for KeyID=0) encryption.	
		63:MAXPHYSADDR	Reserved; must be zero.	
990H	2448	IA32_COPY_STATUS[5]	Status of Most Recent Platform to Local or Local to Platform Copies (R/O)	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		0	IWKEY_COPY_SUCCESSFUL: Status of most recent copy to or from IWKeyBackup	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] = 1))
		63:1	Reserved	
991H	2449	IA32_IWKEYBACKUP_STATUS[4]	Information about IWKeyBackup Register (R/O)	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(07H,0).ECX[23] =1))

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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,0).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,0).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,0).ECX[23] =1))
		63:4	Reserved	
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	
C81H	3201	IA32_L3_QOS_CFG	L3 QOS Configuration (R/W)	If (CPUID.(EAX=10H, ECX=1):ECX.[2] = 1)

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	Enable (R/W) Set 1 to enable L3 CAT masks and COS to operate in Code and Data Prioritization (CDP) mode.	
		63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
C82H	3202	IA32_L2_QOS_CFG	L2 QOS Configuration (R/W)	If (CPUID.(EAX=10H, ECX=2):ECX.[2] = 1)
		0	Enable (R/W) Set 1 to enable L2 CAT masks and COS to operate in Code and Data Prioritization (CDP) mode.	
		63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
C8DH	3213	IA32_QM_EVTSEL	Monitoring Event Select Register (R/W)	If (CPUID.(EAX=07H, ECX=0):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.(EAX= 0FH, ECX=0H).EBX[31:0] +1))
		63:N+32	Reserved	
C8EH	3214	IA32_QM_CTR	Monitoring Counter Register (R/O)	If (CPUID.(EAX=07H, ECX=0):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.	
C8FH	3215	IA32_PQR_ASSOC	Resource Association Register (R/W)	If ((CPUID.(EAX=07H, ECX=0):EBX[12] =1) or (CPUID.(EAX=07H, ECX=0):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.(EAX= 0FH, ECX=0H).EBX[31:0] +1))
		31:N	Reserved	
		63:32	COS (R/W): The class of service (COS) to enforce (on writes); returns the current COS when read.	If (CPUID.(EAX=07H, ECX=0):EBX.[15] = 1)

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
C90H - D8FH	3216 - 3471	Reserved MSR Address Space for CAT Mask Registers	See Section 17.19.4.1, "Enumeration and Detection Support of Cache Allocation Technology".	
C90H	3216	IA32_L3_MASK_0	L3 CAT Mask for COS0 (R/W)	If (CPUID.(EAX=10H, ECX=0H):EBX[1] != 0)
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
C90H+n	3216+n	IA32_L3_MASK_n	L3 CAT Mask for COSn (R/W)	n = CPUID.(EAX=10H, ECX=1H):EDX[15:0]
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
D10H - D4FH	3344 - 3407	Reserved MSR Address Space for L2 CAT Mask Registers	See Section 17.19.4.1, "Enumeration and Detection Support of Cache Allocation Technology".	
D10H	3344	IA32_L2_MASK_0	L2 CAT Mask for COS0 (R/W)	If (CPUID.(EAX=10H, ECX=0H):EBX[2] != 0)
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
D10H+n	3344+n	IA32_L2_MASK_n	L2 CAT Mask for COSn (R/W)	n = CPUID.(EAX=10H, ECX=2H):EDX[15:0]
		31:0	Capacity Bit Mask (R/W)	
		63:32	Reserved	
D90H	3472	IA32_BNDCFGS	Supervisor State of MPX Configuration (R/W)	If (CPUID.(EAX=07H, ECX=0H):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.	
D91H	3473	IA32_COPY_LOCAL_TO_PLATFORM[4]	Copy Local State to Platform State (W)	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		0	IWKeyBackup Copy IWKey to IWKeyBackup	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))
		63:1	Reserved	
D92H	3474	IA32_COPY_PLATFORM_TO_LOCAL[4]	Copy Platform State to Local State (W)	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))
		0	IWKeyBackup Copy IWKeyBackup to IWKey	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.(EAX=07H, ECX=0H).ECX[23] = 1))
		63:1	Reserved	
DA0H	3488	IA32_XSS	Extended Supervisor State Mask (R/W)	If(CPUID.(0DH, 1):EAX.[3] = 1
		7:0	Reserved.	
		8	Trace Packet Configuration State (R/W)	
		10:9	Reserved.	
		11	CET_U State (R/W)	
		12	CET_S State (R/W)	
		13	HDC State (R/W)	
		63:14	Reserved.	
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 14.5.2, "Package level Enabling HDC".	If CPUID.06H:EAX.[13] = 1
		63:1	Reserved	
DB1H	3505	IA32_PM_CTL1	Enable/disable HWP (R/W)	If CPUID.06H:EAX.[13] = 1
		0	HDC_Allow_Block (R/W) Allow/Block this logical processor for package level HDC control. See Section 14.5.3.	If CPUID.06H:EAX.[13] = 1
		63:1	Reserved	
DB2H	3506	IA32_THREAD_STALL	Per-Logical_Processor 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 14.5.4.1.	If CPUID.06H:EAX.[13] = 1
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 ≥ IA32_LBR_DEPTH.DEPTH will #GP.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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 18.1.3.3.	Reset Value: 0
		61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel TSX (CPUID.07H.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 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.EBX.HLE[bit 4]=0 and CPUID.07H.EBX.RTM[bit 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
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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		2	USR When set, allows LBR recording when CPL != 0.	Reset Value: 0
		3	CALL_STACK When set, records branches in call-stack mode. See Section 18.1.2.4.	Reset Value: 0
		15:4	Reserved	Reset Value: 0
		16	COND When set, records taken conditional branches. See Section 18.1.2.3.	
		17	NEAR_REL_JMP When set, records near relative JMPs. See Section 18.1.2.3.	
		18	NEAR_IND_JMP When set, records near indirect JMPs. See Section 18.1.2.3.	
		19	NEAR_REL_CALL When set, records near relative CALLs. See Section 18.1.2.3.	
		20	NEAR_IND_CALL When set, records near indirect CALLs. See Section 18.1.2.3.	
		21	NEAR_RET When set, records near RETs. See Section 18.1.2.3.	
		22	OTHER_BRANCH When set, records other branches. See Section 18.1.2.3.	
		63:23	Reserved	
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.(EAX=01CH,ECX=0):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

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
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 ≥ 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
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 ≥ 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
17D0H	6096	IA32_HW_FEEDBACK_PTR	Hardware Feedback Interface Pointer	If CPUID.06H:EAX.[19] = 1
		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	
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	
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	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
		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.	
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	
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	
4000_0000H - 4000_00FFH		Reserved MSR Address Space	All existing and future processors will not implement MSRs in this range.	
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.	
		11	Execute Disable Bit Enable: IA32_EFER.NXE (R/W)	
		63:12	Reserved	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name / Bit Fields (Former MSR Name)	MSR/Bit Description	Comment
Hex	Decimal			
C000_0081H		IA32_STAR	System Call Target Address (R/W)	If CPUID.80000001:EDX.[29] = 1
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.80000001:EDX.[29] = 1
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.80000001:EDX.[29] = 1
C000_0084H		IA32_FMASK	System Call Flag Mask (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0100H		IA32_FS_BASE	Map of BASE Address of FS (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0101H		IA32_GS_BASE	Map of BASE Address of GS (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0102H		IA32_KERNEL_GS_BASE	Swap Target of BASE Address of GS (R/W)	If CPUID.80000001:EDX.[29] = 1
C000_0103H		IA32_TSC_AUX	Auxiliary TSC (RW)	If CPUID.80000001H: EDX[27] = 1 or CPUID.(EAX=7,ECX=0):ECX[bit 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_MC*i*_STATUS. See Section 15.3.2.3 and Section 15.3.2.4 for more information.
4. MAXPHYADDR is reported by CPUID.80000008H:EAX[7:0].
5. 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 with DisplayFamily_DisplayModel 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 micro-architecture. Processors based on the Enhanced Intel Core microarchitecture have the CPUID signature DisplayFamily_DisplayModel 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		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Unique	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Unique	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination." and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Shared	Platform ID (R) See Table 2-2.
17H	23	MSR_PLATFORM_ID	Shared	Model Specific Platform ID (R)
		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
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, "Local APIC Status and Location" and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	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/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.
		4		Address Parity 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		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		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)
3AH	58	MSR_FEATURE_CONTROL	Unique	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		3	Unique	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.
40H	64	MSR_LASTBRANCH_0_FROM_IP	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 17.5.
41H	65	MSR_LASTBRANCH_1_FROM_IP	Unique	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
42H	66	MSR_LASTBRANCH_2_FROM_IP	Unique	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Unique	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Unique	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.
61H	97	MSR_LASTBRANCH_1_TO_IP	Unique	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Unique	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Unique	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Unique	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	BIOS Update Signature ID (RO) See Table 2-2.
A0H	160	MSR_SMRR_PHYSBASE	Unique	System Management Mode Base Address register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.
		11:0		Reserved
		31:12		PhysBase: SMRR physical Base Address.
		63:32		Reserved
A1H	161	MSR_SMRR_PHYSMASK	Unique	System Management Mode Physical Address Mask register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.
		10:0		Reserved
		11		Valid: Physical address base and range mask are valid.
		31:12		PhysMask: SMRR physical address range mask.
		63:32		Reserved
C1H	193	IA32_PMC0	Unique	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Unique	Performance Counter Register See Table 2-2.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed(RO) This field indicates the intended scaleable bus clock speed for processors based on Intel Core microarchitecture.
		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
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed(RO) This field indicates the intended scaleable bus clock speed for processors based on Enhanced Intel Core microarchitecture.
		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
E7H	231	IA32_MPERF	Unique	Maximum Performance Frequency Clock Count (RW) See Table 2-2.
E8H	232	IA32_APERF	Unique	Actual Performance Frequency Clock Count (RW) See Table 2-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 2-2.
		11	Unique	SMRR Capability Using MSR 0A0H and 0A1H (R)

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	Global Machine Check Status
		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
186H	390	IA32_PERFEVTSEL0	Unique	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
198H	408	MSR_PERF_STATUS	Shared	Current performance status. See Section 14.1.1, “Software Interface For Initiating Performance State Transitions”.
		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.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		63:47		Reserved
199H	409	IA32_PERF_CTL	Unique	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Unique	Thermal Monitor Status (R/W) See Table 2-2.
19DH	413	MSR_THERM2_CTL	Unique	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 are enabled.
		63:16		Reserved
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	Unique	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.
		6:4		Reserved
		7	Shared	Performance Monitoring Available (R) See Table 2-2.
		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.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		10	Shared	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.
		11	Shared	Branch Trace Storage Unavailable (RO) See Table 2-2.
		12	Shared	Processor Event Based Sampling Unavailable (RO) See Table 2-2.
		13	Shared	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.1: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.
		15:14		Reserved
		16	Shared	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Shared	ENABLE MONITOR FSM (R/W) See Table 2-2.
		19	Shared	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.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		20	Shared	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.
		21		Reserved
		22	Shared	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Shared	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Unique	XD Bit Disable (R/W) See Table 2-2.
		36:35		Reserved
		37	Unique	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.
		38	Shared	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.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		39	Unique	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.
		63:40		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Unique	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).
1D9H	473	IA32_DEBUGCTL	Unique	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	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.
1DEH	478	MSR_LER_TO_LIP	Unique	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.
200H	512	IA32_MTRR_PHYSBASE0	Unique	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Unique	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Unique	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Unique	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Unique	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Unique	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Unique	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Unique	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Unique	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Unique	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Unique	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Unique	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Unique	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Unique	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Unique	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Unique	See Table 2-2.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
250H	592	IA32_MTRR_FIX64K_00000	Unique	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Unique	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Unique	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Unique	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Unique	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Unique	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Unique	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Unique	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Unique	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Unique	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Unique	See Table 2-2.
277H	631	IA32_PAT	Unique	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Unique	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Unique	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Unique	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Unique	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
345H	837	MSR_PERF_CAPABILITIES	Unique	RO. This applies to processors that do not support architectural perfmon version 2.
		5:0		LBR Format. See Table 2-2.
		6		PEBS Record Format
		7		PEBSSaveArchRegs. See Table 2-2.
		63:8		Reserved
38DH	909	IA32_FIXED_CTR_CTRL	Unique	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Unique	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STATUS	Unique	See Section 19.6.2.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
38FH	911	MSR_PERF_GLOBAL_CTRL	Unique	See Section 19.6.2.2, "Global Counter Control Facilities."

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Unique	See Section 19.6.2.2, "Global Counter Control Facilities."
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Unique	See Table 2-2. See Section 19.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MC0_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV 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.
408H	1032	IA32_MC2_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV 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.
40CH	1036	IA32_MC4_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC4_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC4_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV 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.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
410H	1040	IA32_MC3_CTL		See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC3_STATUS		See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC3_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV 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.
413H	1043	IA32_MC3_MISC	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.
414H	1044	IA32_MC5_CTL	Unique	Machine Check Error Reporting Register: Controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
415H	1045	IA32_MC5_STATUS	Unique	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.
416H	1046	IA32_MC5_ADDR	Unique	Machine Check Error Reporting Register: Contains the address of the code or data memory location that produced the machine-check error if the ADDRV flag in the IA32_MCi_STATUS register is set.
417H	1047	IA32_MC5_MISC	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.
419H	1045	IA32_MC6_STATUS	Unique	Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 23.
480H	1152	IA32_VMX_BASIC	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTLS	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTLS	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTLS	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
484H	1156	IA32_VMX_ENTRY_CTLS	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Unique	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Unique	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."
107CCH	67532	MSR_EMON_L3_CTR_CTL0	Unique	GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CDH	67533	MSR_EMON_L3_CTR_CTL1	Unique	GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CEH	67534	MSR_EMON_L3_CTR_CTL2	Unique	GSNPQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
107CFH	67535	MSR_EMON_L3_CTR_CTL3	Unique	GSNPQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D0H	67536	MSR_EMON_L3_CTR_CTL4	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D1H	67537	MSR_EMON_L3_CTR_CTL5	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D2H	67538	MSR_EMON_L3_CTR_CTL6	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D3H	67539	MSR_EMON_L3_CTR_CTL7	Unique	FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D8H	67544	MSR_EMON_L3_GL_CTL	Unique	L3/FSB Common Control Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
C000_0080H		IA32_EFER	Unique	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Unique	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Unique	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Unique	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Unique	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Unique	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Unique	Swap Target of BASE Address of GS (R/W) See Table 2-2.

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 with DisplayFamily_DisplayModel of 06_1CH, 06_26H, 06_27H, 06_35H and 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 45 nm and 32 nm Intel Atom® Processor Family

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Shared	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Shared	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_ SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination." and Table 2-2.
10H	16	IA32_TIME_STAMP_ COUNTER	Unique	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Shared	Platform ID (R) See Table 2-2.
17H	23	MSR_PLATFORM_ID	Shared	Model Specific Platform ID (R)
		7:0		Reserved
		12:8		Maximum Qualified Ratio (R) The maximum allowed bus ratio.
		63:13		Reserved
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, "Local APIC Status and Location" and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Shared	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/W) 1 = Enabled; 0 = Disabled Always 0.
		2		Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled Always 0.
		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

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		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)
3AH	58	IA32_FEATURE_CONTROL	Unique	Control Features in Intel 64Processor (R/W) See Table 2-2.
40H	64	MSR_LASTBRANCH_0_FROM_IP	Unique	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 17.5.
41H	65	MSR_LASTBRANCH_1_FROM_IP	Unique	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_LASTBRANCH_2_FROM_IP	Unique	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Unique	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_LASTBRANCH_4_FROM_IP	Unique	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_LASTBRANCH_5_FROM_IP	Unique	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_LASTBRANCH_6_FROM_IP	Unique	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
47H	71	MSR_LASTBRANCH_7_FROM_IP	Unique	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Unique	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.
61H	97	MSR_LASTBRANCH_1_TO_IP	Unique	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Unique	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Unique	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
64H	100	MSR_LASTBRANCH_4_TO_IP	Unique	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
65H	101	MSR_LASTBRANCH_5_TO_IP	Unique	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
66H	102	MSR_LASTBRANCH_6_TO_IP	Unique	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
67H	103	MSR_LASTBRANCH_7_TO_IP	Unique	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
79H	121	IA32_BIOS_UPDT_TRIG	Shared	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	BIOS Update Signature ID (RO) See Table 2-2.
C1H	193	IA32_PMC0	Unique	Performance counter register See Table 2-2.
C2H	194	IA32_PMC1	Unique	Performance Counter Register See Table 2-2.
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed(RO) This field indicates the intended scaleable bus clock speed for processors based on Intel Atom microarchitecture.
		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

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
E7H	231	IA32_MPERF	Unique	Maximum Performance Frequency Clock Count (RW) See Table 2-2.
E8H	232	IA32_APERF	Unique	Actual Performance Frequency Clock Count (RW) See Table 2-2.
FEH	254	IA32_MTRRCAP	Shared	Memory Type Range Register (R) See Table 2-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	Control Register 3 Used to configure the L2 Cache.
		0		L2 Hardware Enabled (RO) 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 (RO) 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	Global Machine Check Status
		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 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		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
186H	390	IA32_PERFEVTSEL0	Unique	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
198H	408	MSR_PERF_STATUS	Shared	Performance Status
		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
199H	409	IA32_PERF_CTL	Unique	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Unique	Thermal Monitor Status (R/W) See Table 2-2.
19DH	413	MSR_THERM2_CTL	Shared	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 are enabled.
		63:17		Reserved
1A0H	416	IA32_MISC_ENABLE	Unique	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.

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		2:1		Reserved
		3	Unique	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.
		6:4		Reserved
		7	Shared	Performance Monitoring Available (R) See Table 2-2.
		8		Reserved
		9		Reserved
		10	Shared	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.
		11	Shared	Branch Trace Storage Unavailable (RO) See Table 2-2.
		12	Shared	Processor Event Based Sampling Unavailable (RO) See Table 2-2.
		13	Shared	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.1: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.
		15:14		Reserved
		16	Shared	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Shared	ENABLE MONITOR FSM (R/W) See Table 2-2.
		19		Reserved

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
		20	Shared	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.
		21		Reserved
		22	Unique	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Shared	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Unique	XD Bit Disable (R/W) See Table 2-2.
		63:35		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Unique	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).
1D9H	473	IA32_DEBUGCTL	Unique	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	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.
1DEH	478	MSR_LER_TO_LIP	Unique	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.
200H	512	IA32_MTRR_PHYSBASE0	Shared	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Shared	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Shared	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Shared	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Shared	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Shared	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Shared	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Shared	See Table 2-2.

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
208H	520	IA32_MTRR_PHYSBASE4	Shared	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Shared	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Shared	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Shared	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Shared	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Shared	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Shared	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Shared	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Shared	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Shared	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Shared	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Shared	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Shared	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Shared	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Shared	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Shared	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Shared	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Shared	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Shared	See Table 2-2.
277H	631	IA32_PAT	Unique	See Table 2-2.
309H	777	IA32_FIXED_CTR0	Unique	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Unique	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Unique	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Shared	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Unique	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Unique	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Unique	See Table 2-2. See Section 19.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0 (R/W)

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
400H	1024	IA32_MC0_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV 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.
40CH	1036	IA32_MC3_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV 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.
410H	1040	IA32_MC4_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC4_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV 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.
480H	1152	IA32_VMX_BASIC	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
481H	1153	IA32_VMX_PINBASED_CTLS	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTLS	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTLS	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Unique	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Unique	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."
C000_0080H		IA32_EFER	Unique	Extended Feature Enables See Table 2-2.

Table 2-4. MSRs in 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address		Register Name / Bit Fields	Shared/ Unique	Bit Description
Hex	Dec			
C000_ 0081H		IA32_STAR	Unique	System Call Target Address (R/W) See Table 2-2.
C000_ 0082H		IA32_LSTAR	Unique	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_ 0084H		IA32_FMASK	Unique	System Call Flag Mask (R/W) See Table 2-2.
C000_ 0100H		IA32_FS_BASE	Unique	Map of BASE Address of FS (R/W) See Table 2-2.
C000_ 0101H		IA32_GS_BASE	Unique	Map of BASE Address of GS (R/W) See Table 2-2.
C000_ 0102H		IA32_KERNEL_GS_BASE	Unique	Swap Target of BASE Address of GS (R/W) See Table 2-2.

Table 2-5 lists model-specific registers (MSRs) that are specific to Intel Atom® processor with the CPUID signature with DisplayFamily_DisplayModel of 06_27H.

Table 2-5. MSRs Supported by Intel Atom® Processors with CPUID Signature 06_27H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3F8H	1016	MSR_PKG_C2_RESIDENCY	Package	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.
		63:0	Package	Package C2 Residency Counter (R/O) Time that this package is in processor-specific C2 states since last reset. Counts at 1 Mhz frequency.
3F9H	1017	MSR_PKG_C4_RESIDENCY	Package	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.
		63:0	Package	Package C4 Residency Counter. (R/O) Time that this package is in processor-specific C4 states since last reset. Counts at 1 Mhz frequency.
3FAH	1018	MSR_PKG_C6_RESIDENCY	Package	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.
		63:0	Package	Package C6 Residency Counter. (R/O) Time that this package is in processor-specific C6 states since last reset. Counts at 1 Mhz frequency.

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 with DisplayFamily_DisplayModel of 06_37H, 06_4AH, 06_4DH, 06_5AH, and 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 leaf 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 the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Module	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Module	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Core	See Section 8.10.5, "Monitor/Mwait Address Range Determination." and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Core	See Section 17.17, "Time-Stamp Counter," and Table 2-2.
1BH	27	IA32_APIC_BASE	Core	See Section 10.4.4, "Local APIC Status and Location," and Table 2-2.
2AH	42	MSR_EBL_CR_POWERON	Module	Processor Hard Power-On Configuration (R/W) Writes ignored.
		63:0		Reserved
34H	52	MSR_SMI_COUNT	Core	SMI Counter (R/O)
		31:0		SMI Count (R/O) Running count of SMI events since last RESET.
		63:32		Reserved
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Core	BIOS Update Signature ID (RO) See Table 2-2.
C1H	193	IA32_PMC0	Core	Performance counter register See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C2H	194	IA32_PMC1	Core	Performance Counter Register See Table 2-2.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Module	Power Management IO Redirection in C-state (R/W) See http://biosbits.org.
		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[bit10]: 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
E7H	231	IA32_MPERF	Core	Maximum Performance Frequency Clock Count (RW) See Table 2-2.
E8H	232	IA32_APERF	Core	Actual Performance Frequency Clock Count (RW) See Table 2-2.
FEH	254	IA32_MTRRCAP	Core	Memory Type Range Register (R) See Table 2-2.
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		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
174H	372	IA32_SYSENTER_CS	Core	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Core	See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
176H	374	IA32_SYSENTER_EIP	Core	See Table 2-2.
179H	377	IA32_MCG_CAP	Core	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Core	Global Machine Check Status
		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
186H	390	IA32_PERFEVTSEL0	Core	See Table 2-2.
		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
187H	391	IA32_PERFEVTSEL1	Core	See Table 2-2.
198H	408	IA32_PERF_STATUS	Module	See Table 2-2.
199H	409	IA32_PERF_CTL	Core	See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
19AH	410	IA32_CLOCK_MODULATION	Core	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Core	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		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
1A6H	422	MSR_OFFCORE_RSP_0	Module	Offcore Response Event Select Register (R/W)
1A7H	423	MSR_OFFCORE_RSP_1	Module	Offcore Response Event Select Register (R/W)
1B0H	432	IA32_ENERGY_PERF_BIAS	Core	See Table 2-2.
1D9H	473	IA32_DEBUGCTL	Core	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Core	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.
1DEH	478	MSR_LER_TO_LIP	Core	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.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 2-2.
277H	631	IA32_PAT	Core	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Core	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Core	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Core	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Core	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Core	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Core	Fixed-Function-Counter Control Register (R/W) See Table 2-2.

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
38FH	911	IA32_PERF_GLOBAL_CTRL	Core	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
400H	1024	IA32_MC0_CTL	Module	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Module	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Module	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Module	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Module	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Module	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Module	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Module	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV 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.
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV 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.
410H	1040	IA32_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
412H	1042	IA32_MC4_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV 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.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
416H	1046	IA32_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV 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.
480H	1152	IA32_VMX_BASIC	Core	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTLS	Core	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTLS	Core	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTLS	Core	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Core	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Core	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Core	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
487H	1159	IA32_VMX_CR0_FIXED1	Core	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Core	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Core	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Core	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Core	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Core	Capability Reporting Register of EPT and VPID (R/O) See Table 2-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	Core	Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	Core	Capability Reporting Register of Primary Processor-based VM-Execution Flex Controls (R/O) See Table 2-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	Core	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	Core	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2
491H	1169	IA32_VMX_FMFUNC	Core	Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2
4C1H	1217	IA32_A_PMC0	Core	See Table 2-2.
4C2H	1218	IA32_A_PMC1	Core	See Table 2-2.
600H	1536	IA32_DS_AREA	Core	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."

Table 2-6. MSRs Common to the Silvermont Microarchitecture and Newer Microarchitectures for Intel Atom® Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
660H	1632	MSR_CORE_C1_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
6E0H	1760	IA32_TSC_DEADLINE	Core	TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.
C000_0080H		IA32_EFER	Core	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Core	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Core	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Core	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Core	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Core	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Core	Swap Target of BASE Address of GS (R/W) See Table 2-2.
C000_0103H		IA32_TSC_AUX	Core	AUXILIARY TSC Signature (R/W) See Table 2-2

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		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
17H	23	MSR_PLATFORM_ID	Module	Model Specific Platform ID (R)
		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

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Core	Control Features in Intel 64Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Reserved
		2		Enable VMX outside SMX operation (R/WL)
40H	64	MSR_LASTBRANCH_0_FROM_IP	Core	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 17.5 and record format in Section 17.4.8.1.
41H	65	MSR_LASTBRANCH_1_FROM_IP	Core	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_LASTBRANCH_2_FROM_IP	Core	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_LASTBRANCH_3_FROM_IP	Core	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_LASTBRANCH_4_FROM_IP	Core	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_LASTBRANCH_5_FROM_IP	Core	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_LASTBRANCH_6_FROM_IP	Core	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
47H	71	MSR_LASTBRANCH_7_FROM_IP	Core	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_LASTBRANCH_0_TO_IP	Core	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.
61H	97	MSR_LASTBRANCH_1_TO_IP	Core	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
62H	98	MSR_LASTBRANCH_2_TO_IP	Core	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
63H	99	MSR_LASTBRANCH_3_TO_IP	Core	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
64H	100	MSR_LASTBRANCH_4_TO_IP	Core	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
65H	101	MSR_LASTBRANCH_5_TO_IP	Core	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
66H	102	MSR_LASTBRANCH_6_TO_IP	Core	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
67H	103	MSR_LASTBRANCH_7_TO_IP	Core	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information: Contains power management and other model specific features enumeration. See http://biosbits.org.
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * Scalable Bus Frequency.
		63:16		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Module	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.
		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-sate 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
11EH	281	MSR_BBL_CR_CTL3	Module	Control Register 3 Used to configure the L2 Cache.
		0		L2 Hardware Enabled (RO) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.
		7:1		Reserved

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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 (RO) 0 = L2 Present. 1 = L2 Not Present.
		63:24		Reserved
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Core	Fast-Strings Enable See Table 2-2.
		2:1		Reserved
		3	Module	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.
		6:4		Reserved
		7	Core	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Core	Branch Trace Storage Unavailable (RO) See Table 2-2.
		12	Core	Processor Event Based Sampling Unavailable (RO) See Table 2-2.
		15:13		Reserved
		16	Module	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Core	ENABLE MONITOR FSM (R/W) See Table 2-2.
		21:19		Reserved
		22	Core	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Module	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Core	XD Bit Disable (R/W) See Table 2-2.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		37:35		Reserved
		38	Module	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.
		63:39		Reserved
1C8H	456	MSR_LBR_SELECT	Core	Last Branch Record Filtering Select Register (R/W) See Section 17.9.2, "Filtering of Last Branch Records."
		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
1C9H	457	MSR_LASTBRANCH_TOS	Core	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.
38EH	910	IA32_PERF_GLOBAL_STATUS	Core	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Core	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	See Table 2-2. See Section 19.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS for precise event on IA32_PMC0 (R/W)
3FAH	1018	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	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.
		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 model-specific registers (MSRs) that are specific to Intel Atom® processor E3000 Series (CPUID signature with DisplayFamily_DisplayModel of 06_37H) and Intel Atom processors (CPUID signatures with DisplayFamily_DisplayModel of 06_4AH, 06_5AH, 06_5DH).

Table 2-8. Specific MSRs Supported by Intel Atom® Processors with CPUID Signatures 06_37H, 06_4AH, 06_5AH, 06_5DH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Module	Scaleable Bus Speed(RO) This field indicates the intended scaleable bus clock speed for processors based on Silvermont microarchitecture.
		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
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 14.10.1, "RAPL Interfaces."
		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.

Table 2-8. Specific MSRs Supported by Intel Atom® Processors with CPUID Signatures 06_37H, 06_4AH, 06_5AH, 06_5DH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15:13		Reserved
		19:16		Time Unit The value is 0000b, indicating time unit is in one second.
		63:20		Reserved
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W)
		14:0		Package Power Limit #1 (R/W) See Section 14.10.3, "Package RAPL Domain." and MSR_RAPL_POWER_UNIT in Table 2-8.
		15		Enable Power Limit #1 (R/W) See Section 14.10.3, "Package RAPL Domain."
		16		Package Clamping Limitation #1 (R/W) See Section 14.10.3, "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
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 14.10.3, "Package RAPL Domain." and MSR_RAPL_POWER_UNIT in Table 2-8.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains." and MSR_RAPL_POWER_UNIT in Table 2-8.

Table 2-9 lists model-specific registers (MSRs) that are specific to Intel Atom® processor E3000 Series (CPUID signature with DisplayFamily_DisplayModel of 06_37H).

Table 2-9. Specific MSRs Supported by Intel Atom® Processor E3000 Series with CPUID Signature 06_37H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
668H	1640	MSR_CC6_DEMOTION_POLICY_CONFIG	Package	Core C6 Demotion Policy Config MSR
		63:0		Controls per-core C6 demotion policy. Writing a value of 0 disables core level HW demotion policy.
669H	1641	MSR_MC6_DEMOTION_POLICY_CONFIG	Package	Module C6 Demotion Policy Config MSR
		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.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	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.

Table 2-9. Specific MSRs Supported by Intel Atom® Processor E3000 Series (Contd.)with CPUID Signature 06_37H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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 with DisplayFamily_DisplayModel of 06_4DH).

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series with CPUID Signature 06_4DH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	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.
		1		Reserved
		2	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		63:3		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode (RW)
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.
		55:48	Package	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.
		63:56	Package	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 14.10.1, "RAPL Interfaces."

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series (Contd.)with CPUID Signature

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 14.10.3, "Package RAPL Domain."
66EH	1646	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameter (R/0)
		14:0		Thermal Spec Power (R/0) 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 with DisplayFamily_DisplayModel including 06_4CH; see Table 2-1.

Table 2-11. MSRs in Intel Atom® Processors Based on the Airmont Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Module	Scaleable Bus Speed(RO) This field indicates the intended scaleable bus clock speed for processors based on Airmont microarchitecture.

Table 2-11. MSRs in Intel Atom® Processors Based on the Airmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Module	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.
		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/WO) When set, locks bits 15:0 of this register until next reset.
		63:16		Reserved
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Module	Power Management IO Redirection in C-state (R/W) See http://biosbits.org.
		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 2-11. MSRs in Intel Atom® Processors Based on the Airmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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[bit10]: 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
638H	1592	MSR_PP0_POWER_LIMIT	Package	PP0 RAPL Power Limit Control (R/W)
		14:0		PP0 Power Limit #1 (R/W) See Section 14.10.4, "PP0/PP1 RAPL Domains" and MSR_RAPL_POWER_UNIT in Table 2-8.
		15		Enable Power Limit #1 (R/W) See Section 14.10.4, "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 with DisplayFamily_DisplayModel 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 leaf 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 the Goldmont Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
17H	23	MSR_PLATFORM_ID	Module	Model Specific Platform ID (R)
		49:0		Reserved
		52:50		See Table 2-2.
		63:33		Reserved
3AH	58	IA32_FEATURE_CONTROL	Core	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		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
3BH	59	IA32_TSC_ADJUST	Core	Per-Core TSC ADJUST (R/W) See Table 2-2.
C3H	195	IA32_PMC2	Core	Performance Counter Register See Table 2-2.
C4H	196	IA32_PMC3	Core	Performance Counter Register See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	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.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29	Package	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.
		30	Package	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.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	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.
		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.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:16		Reserved
17DH	381	MSR_SMM_MCA_CAP	Core	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability enhancement. Accessible only while in SMM.
		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
188H	392	IA32_PERFEVTSEL2	Core	See Table 2-2.
189H	393	IA32_PERFEVTSEL3	Core	See Table 2-2.
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Core	Fast-Strings Enable See Table 2-2.
		2:1		Reserved
		3	Package	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.
		6:4		Reserved
		7	Core	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Core	Branch Trace Storage Unavailable (RO) See Table 2-2.
		12	Core	Processor Event Based Sampling Unavailable (RO) See Table 2-2.
		15:13		Reserved
		16	Package	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Core	ENABLE MONITOR FSM (R/W) See Table 2-2.
		21:19		Reserved
		22	Core	Limit CPUID Maxval (R/W) See Table 2-2.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		23	Package	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Core	XD Bit Disable (R/W) See Table 2-2.
		37:35		Reserved
		38	Package	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.
		63:39		Reserved
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	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.
		1		Reserved
		2	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		63:3		Reserved
1AAH	426	MSR_MISC_PWR_MGMT	Package	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.
		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

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode by Core Groups (RW) 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.
		7:0	Package	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.
		15:8	Package	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.
		23:16	Package	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.
		31:24	Package	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.
		39:32	Package	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.
		47:40	Package	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.
		55:48	Package	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.
		63:56	Package	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.
1AEH	430	MSR_TURBO_GROUP_CORECNT	Package	Group Size of Active Cores for Turbo Mode Operation (RW) Writes of 0 threshold is ignored.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7:0	Package	Group 0 Core Count Threshold Maximum number of active cores to operate under the Group 0 Max Turbo Ratio limit.
		15:8	Package	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.
		23:16	Package	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.
		31:24	Package	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.
		39:32	Package	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.
		47:40	Package	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.
		55:48	Package	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.
		63:56	Package	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.
1C8H	456	MSR_LBR_SELECT	Core	Last Branch Record Filtering Select Register (R/W) See Section 17.9.2, "Filtering of Last Branch Records."
		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 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		8		FAR_BRANCH
		9		EN_CALL_STACK
		63:10		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Core	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.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register. See http://biosbits.org.
		0		Reserved
		1	Package	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).
		63:2		Reserved
210H	528	IA32_MTRR_PHYSBASE8	Core	See Table 2-2.
211H	529	IA32_MTRR_PHYSMASK8	Core	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Core	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Core	See Table 2-2.
280H	640	IA32_MC0_CTL2	Module	See Table 2-2.
281H	641	IA32_MC1_CTL2	Module	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Module	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	See Table 2-2.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
300H	768	MSR_SGXOWNEREPOCH0	Package	Lower 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.
301H	769	MSR_SGXOWNEREPOCH1	Package	Upper 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.
38EH	910	IA32_PERF_GLOBAL_STATUS	Core	See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."
		0		Ovf_PMC0

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
		57:56		Reserved
		58		LBR_Frz.
		59		CTR_Frz.
		60		ASCI
		61		Ovf_Uncore
		62		Ovf_BufDSSAVE
		63		CondChgd
390H	912	IA32_PERF_GLOBAL_STATUS_RESET	Core	See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."
		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.
391H	913	IA32_PERF_GLOBAL_STATUS_SET	Core	See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
		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
392H	914	IA32_PERF_GLOBAL_INUSE	Core	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	See Table 2-2. See Section 19.6.2.4, "Processor Event Based Sampling (PEBS)."
		0		Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0. (R/W)
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
406H	1030	IA32_MC1_ADDR	Module	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV 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.
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
41AH	1050	IA32_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
4C3H	1219	IA32_A_PMC2	Core	See Table 2-2.
4C4H	1220	IA32_A_PMC3	Core	See Table 2-2.
4E0H	1248	MSR_SMM_FEATURE_CONTROL	Package	Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.
		0		Lock (SMM-RWO) 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
4E2H	1250	MSR_SMM_DELAYED	Package	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.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
4E3H	1251	MSR_SMM_BLOCKED	Package	SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.
		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.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
500H	1280	IA32_SGX_SVN_STATUS	Core	Status and SVN Threshold of SGX Support for ACM (RO)
		0		Lock See Section 38.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		15:1		Reserved
		23:16		SGX_SVN_SINIT See Section 38.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		63:24		Reserved
560H	1376	IA32_RTIT_OUTPUT_BASE	Core	Trace Output Base Register (R/W) See Table 2-2.
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Core	Trace Output Mask Pointers Register (R/W) See Table 2-2.
570H	1392	IA32_RTIT_CTL	Core	Trace Control Register (R/W)
		0		TraceEn
		1		CYCEn
		2		OS
		3		User
		6:4		Reserved, must be zero.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7		CR3 filter
		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
		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.
571H	1393	IA32_RTIT_STATUS	Core	Tracing Status Register (R/W)
		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.
572H	1394	IA32_RTIT_CR3_MATCH	Core	Trace Filter CR3 Match Register (R/W)
		4:0		Reserved
		63:5		CR3[63:5] value to match.
580H	1408	IA32_RTIT_ADDR0_A	Core	Region 0 Start Address (R/W)
		63:0		See Table 2-2.
581H	1409	IA32_RTIT_ADDR0_B	Core	Region 0 End Address (R/W)
		63:0		See Table 2-2.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
582H	1410	IA32_RTIT_ADDR1_A	Core	Region 1 Start Address (R/W)
		63:0		See Table 2-2.
583H	1411	IA32_RTIT_ADDR1_B	Core	Region 1 End Address (R/W)
		63:0		See Table 2-2.
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 14.10.1, "RAPL Interfaces."
		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
60AH	1546	MSR_PKGC3_IRTL	Package	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.
		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-sate management.
		63:16		Reserved

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
60BH	1547	MSR_PKGC_IRTL1	Package	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.
		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.
		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-sate management.
		63:16		Reserved
60CH	1548	MSR_PKGC_IRTL2	Package	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.
		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-sate management.
		63:16		Reserved
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 14.10.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 14.10.3, "Package RAPL Domain."
613H	1555	MSR_PKG_PERF_STATUS	Package	PKG Perf Status (R/O) See Section 14.10.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameters (R/W)
		14:0		Thermal Spec Power (R/W) See Section 14.10.3, "Package RAPL Domain."
		15		Reserved
		30:16		Minimum Power (R/W) See Section 14.10.3, "Package RAPL Domain."
		31		Reserved
		46:32		Maximum Power (R/W) See Section 14.10.3, "Package RAPL Domain."
		47		Reserved
		54:48		Maximum Time Window (R/W) Specified by 2^Y * (1.0 + Z/4.0) * 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
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
632H	1586	MSR_PKG_C10_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		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.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	PP1 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W)
		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
64FH	1615	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		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.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
		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.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Core	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 17.6 and record format in Section 17.4.8.1.
		0:47		From Linear Address (R/W)
		62:48		Signed extension of bits 47:0.
		63		Mispred
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Core	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Core	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Core	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Core	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Core	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Core	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Core	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Core	Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Core	Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Core	Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Core	Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Core	Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Core	Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Core	Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Core	Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
690H	1680	MSR_LASTBRANCH_16_FROM_IP	Core	Last Branch Record 16 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
691H	1681	MSR_LASTBRANCH_17_FROM_IP	Core	Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
692H	1682	MSR_LASTBRANCH_18_FROM_IP	Core	Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
693H	1683	MSR_LASTBRANCH_19_FROM_IP	Core	Last Branch Record 19From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
694H	1684	MSR_LASTBRANCH_20_FROM_IP	Core	Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
695H	1685	MSR_LASTBRANCH_21_FROM_IP	Core	Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
696H	1686	MSR_LASTBRANCH_22_FROM_IP	Core	Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
697H	1687	MSR_LASTBRANCH_23_FROM_IP	Core	Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
698H	1688	MSR_LASTBRANCH_24_FROM_IP	Core	Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
699H	1689	MSR_LASTBRANCH_25_FROM_IP	Core	Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69AH	1690	MSR_LASTBRANCH_26_FROM_IP	Core	Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69BH	1691	MSR_LASTBRANCH_27_FROM_IP	Core	Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69CH	1692	MSR_LASTBRANCH_28_FROM_IP	Core	Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69DH	1693	MSR_LASTBRANCH_29_FROM_IP	Core	Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69EH	1694	MSR_LASTBRANCH_30_FROM_IP	Core	Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69FH	1695	MSR_LASTBRANCH_31_FROM_IP	Core	Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Core	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 17.6.
		0:47		Target Linear Address (R/W)
		63:48		Elapsed cycles from last update to the LBR.
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Core	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Core	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Core	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Core	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Core	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Core	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Core	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Core	Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Core	Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Core	Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Core	Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Core	Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Core	Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Core	Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Core	Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D0H	1744	MSR_LASTBRANCH_16_TO_IP	Core	Last Branch Record 16 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D1H	1745	MSR_LASTBRANCH_17_TO_IP	Core	Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D2H	1746	MSR_LASTBRANCH_18_TO_IP	Core	Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D3H	1747	MSR_LASTBRANCH_19_TO_IP	Core	Last Branch Record 19To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D4H	1748	MSR_LASTBRANCH_20_TO_IP	Core	Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D5H	1749	MSR_LASTBRANCH_21_TO_IP	Core	Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D6H	1750	MSR_LASTBRANCH_22_TO_IP	Core	Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D7H	1751	MSR_LASTBRANCH_23_TO_IP	Core	Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D8H	1752	MSR_LASTBRANCH_24_TO_IP	Core	Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D9H	1753	MSR_LASTBRANCH_25_TO_IP	Core	Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DAH	1754	MSR_LASTBRANCH_26_TO_IP	Core	Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6DBH	1755	MSR_LASTBRANCH_27_TO_IP	Core	Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DCH	1756	MSR_LASTBRANCH_28_TO_IP	Core	Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DDH	1757	MSR_LASTBRANCH_29_TO_IP	Core	Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DEH	1758	MSR_LASTBRANCH_30_TO_IP	Core	Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DFH	1759	MSR_LASTBRANCH_31_TO_IP	Core	Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
802H	2050	IA32_X2APIC_APICID	Core	x2APIC ID register (R/O)
803H	2051	IA32_X2APIC_VERSION	Core	x2APIC Version register (R/O)
808H	2056	IA32_X2APIC_TPR	Core	x2APIC Task Priority register (R/W)
80AH	2058	IA32_X2APIC_PPR	Core	x2APIC Processor Priority register (R/O)
80BH	2059	IA32_X2APIC_EOI	Core	x2APIC EOI register (W/O)
80DH	2061	IA32_X2APIC_LDR	Core	x2APIC Logical Destination register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Core	x2APIC Spurious Interrupt Vector register (R/W)
810H	2064	IA32_X2APIC_ISR0	Core	x2APIC In-Service register bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Core	x2APIC In-Service register bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Core	x2APIC In-Service register bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Core	x2APIC In-Service register bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Core	x2APIC In-Service register bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Core	x2APIC In-Service register bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Core	x2APIC In-Service register bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Core	x2APIC In-Service register bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Core	x2APIC Trigger Mode register bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Core	x2APIC Trigger Mode register bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Core	x2APIC Trigger Mode register bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Core	x2APIC Trigger Mode register bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Core	x2APIC Trigger Mode register bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Core	x2APIC Trigger Mode register bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Core	x2APIC Trigger Mode register bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Core	x2APIC Trigger Mode register bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Core	x2APIC Interrupt Request register bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Core	x2APIC Interrupt Request register bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Core	x2APIC Interrupt Request register bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Core	x2APIC Interrupt Request register bits [127:96] (R/O)

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
824H	2084	IA32_X2APIC_IRR4	Core	x2APIC Interrupt Request register bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Core	x2APIC Interrupt Request register bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Core	x2APIC Interrupt Request register bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Core	x2APIC Interrupt Request register bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Core	x2APIC Error Status register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Core	x2APIC LVT Corrected Machine Check Interrupt register (R/W)
830H	2096	IA32_X2APIC_ICR	Core	x2APIC Interrupt Command register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Core	x2APIC LVT Timer Interrupt register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Core	x2APIC LVT Thermal Sensor Interrupt register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Core	x2APIC LVT Performance Monitor register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Core	x2APIC LVT LINT0 register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Core	x2APIC LVT LINT1 register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Core	x2APIC LVT Error register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Core	x2APIC Initial Count register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Core	x2APIC Current Count register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Core	x2APIC Divide Configuration register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Core	x2APIC Self IPI register (W/O)
C8FH	3215	IA32_PQR_ASSOC	Core	Resource Association Register (R/W)
		31:0		Reserved
		33:32		COS (R/W)
		63: 34		Reserved
D10H	3344	IA32_L2_QOS_MASK_0	Module	L2 Class Of Service Mask - COS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.
		0:7		CBM: Bit vector of available L2 ways for COS 0 enforcement.
		63:8		Reserved
D11H	3345	IA32_L2_QOS_MASK_1	Module	L2 Class Of Service Mask - COS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.
		0:7		CBM: Bit vector of available L2 ways for COS 0 enforcement.
		63:8		Reserved
D12H	3346	IA32_L2_QOS_MASK_2	Module	L2 Class Of Service Mask - COS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.
		0:7		CBM: Bit vector of available L2 ways for COS 0 enforcement.

Table 2-12. MSRs in Intel Atom® Processors Based on the Goldmont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:8		Reserved
D13H	3347	IA32_L2_QOS_MASK_3	Package	L2 Class Of Service Mask - COS 3 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.
		0:19		CBM: Bit vector of available L2 ways for COS 3 enforcement.
		63:20		Reserved
D90H	3472	IA32_BNDCFGS	Core	See Table 2-2.
DA0H	3488	IA32_XSS	Core	See Table 2-2.
See Table 2-6, and Table 2-12 for MSR definitions applicable to processors with CPUID signature 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 with DisplayFamily_DisplayModel 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 supersede 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 leaf 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 the Goldmont Plus Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Core	Control Features in Intel 64Processor (R/W) See Table 2-2.
		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.(EAX=07H, ECX=0H): ECX[30] = 1.
		18		SGX global functions enable (R/WL)

Table 2-13. MSRs in Intel Atom® Processors Based on the Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:19		Reserved
8CH	140	IA32_SGXLEPUBKEYHASH0	Core	See Table 2-2.
8DH	141	IA32_SGXLEPUBKEYHASH1	Core	See Table 2-2.
8EH	142	IA32_SGXLEPUBKEYHASH2	Core	See Table 2-2.
8FH	143	IA32_SGXLEPUBKEYHASH3	Core	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	(R/W) See Table 2-2. See Section 19.6.2.4, "Processor Event Based Sampling (PEBS)."
		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
570H	1392	IA32_RTIT_CTL	Core	Trace Control Register (R/W)
		0		TraceEn
		1		CYCEn
		2		OS
		3		User
		4		PwrEvtEn
		5		FUPonPTW
		6		FabricEn
		7		CR3 filter
		8		ToPA Writing 0 will #GP if also setting TraceEn.
		9		MTCEn
		10		TSCEn

Table 2-13. MSRs in Intel Atom® Processors Based on the Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Core	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 17.7, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture."
681H - 69FH	1665 - 1695	MSR_LASTBRANCH_*i*_FROM_IP	Core	Last Branch Record *i* From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP; *i* = 1-31.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Core	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 17.7, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture."
6C1H - 6DFH	1729 - 1759	MSR_LASTBRANCH_*i*_TO_IP	Core	Last Branch Record *i* To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP; *i* = 1-31.
DC0H	3520	MSR_LASTBRANCH_INFO_0	Core	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 17.9.1, "LBR Stack."
DC1H	3521	MSR_LASTBRANCH_INFO_1	Core	Last Branch Record 1 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.

Table 2-13. MSRs in Intel Atom® Processors Based on the Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DC2H	3522	MSR_LASTBRANCH_INFO_2	Core	Last Branch Record 2 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC3H	3523	MSR_LASTBRANCH_INFO_3	Core	Last Branch Record 3 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC4H	3524	MSR_LASTBRANCH_INFO_4	Core	Last Branch Record 4 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC5H	3525	MSR_LASTBRANCH_INFO_5	Core	Last Branch Record 5 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC6H	3526	MSR_LASTBRANCH_INFO_6	Core	Last Branch Record 6 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC7H	3527	MSR_LASTBRANCH_INFO_7	Core	Last Branch Record 7 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC8H	3528	MSR_LASTBRANCH_INFO_8	Core	Last Branch Record 8 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DC9H	3529	MSR_LASTBRANCH_INFO_9	Core	Last Branch Record 9 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCAH	3530	MSR_LASTBRANCH_INFO_10	Core	Last Branch Record 10 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCBH	3531	MSR_LASTBRANCH_INFO_11	Core	Last Branch Record 11 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCCH	3532	MSR_LASTBRANCH_INFO_12	Core	Last Branch Record 12 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCDH	3533	MSR_LASTBRANCH_INFO_13	Core	Last Branch Record 13 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCEH	3534	MSR_LASTBRANCH_INFO_14	Core	Last Branch Record 14 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DCFH	3535	MSR_LASTBRANCH_INFO_15	Core	Last Branch Record 15 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD0H	3536	MSR_LASTBRANCH_INFO_16	Core	Last Branch Record 16 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD1H	3537	MSR_LASTBRANCH_INFO_17	Core	Last Branch Record 17 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD2H	3538	MSR_LASTBRANCH_INFO_18	Core	Last Branch Record 18 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD3H	3539	MSR_LASTBRANCH_INFO_19	Core	Last Branch Record 19 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD4H	3520	MSR_LASTBRANCH_INFO_20	Core	Last Branch Record 20 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.

Table 2-13. MSRs in Intel Atom® Processors Based on the Goldmont Plus Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DD5H	3521	MSR_LASTBRANCH_INFO_21	Core	Last Branch Record 21 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD6H	3522	MSR_LASTBRANCH_INFO_22	Core	Last Branch Record 22 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD7H	3523	MSR_LASTBRANCH_INFO_23	Core	Last Branch Record 23 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD8H	3524	MSR_LASTBRANCH_INFO_24	Core	Last Branch Record 24 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DD9H	3525	MSR_LASTBRANCH_INFO_25	Core	Last Branch Record 25 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDAH	3526	MSR_LASTBRANCH_INFO_26	Core	Last Branch Record 26 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDBH	3527	MSR_LASTBRANCH_INFO_27	Core	Last Branch Record 27 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDCH	3528	MSR_LASTBRANCH_INFO_28	Core	Last Branch Record 28 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDDH	3529	MSR_LASTBRANCH_INFO_29	Core	Last Branch Record 29 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDEH	3530	MSR_LASTBRANCH_INFO_30	Core	Last Branch Record 30 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
DDFH	3531	MSR_LASTBRANCH_INFO_31	Core	Last Branch Record 31 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.
See Table 2-6, Table 2-12 and Table 2-13 for MSR definitions applicable to processors with CPUID signature 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 with DisplayFamily_DisplayModel 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 supersede 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 leaf 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 the Tremont Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
33H	51	MSR_MEMORY_CTRL	Core	Memory Control Register
		28:0		Reserved.
		29		Enable #AC(0) exception for split locked accesses: Cause #AC(0) exception for split locked access at all CPL irrespective of CR0.AM or EFLAGS.AC. If bits 29 and 31 are both set, bit 29 takes precedence.
		30		Reserved.
		31		Reserved.
CFH	207	IA32_CORE_CAPABILITIES	Core	IA32 Core Capabilities Register If CPUID.(EAX=07H, ECX=0):EDX[30] = 1.
		4:0		Reserved.
		5		Bit 29 of MSR_MEMORY_CTRL (address 33H) supported.
		63:6		Reserved.
2A0H	672	MSR_PRMRR_BASE_0	Core	Processor Reserved Memory Range Register - Physical Base Control Register (R/W)
		2:0		MEMTYPE: PRMRR BASE Memory Type.
		3		CONFIGURED: PRMRR BASE Configured.
		11:4		Reserved.
		51:12		BASE: PRMRR Base Address.
		63:52		Reserved.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Core	(R/W) See Table 2-2. See Section 19.6.2.4, "Processor Event Based Sampling (PEBS)".
		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.0.EBX[25] are set. 10B: Reserved 11B: Reserved

Table 2-14. MSRs in Intel Atom® Processors Based on the Tremont Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63		Reserved.
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.
14C1H - 14C4H	5313 - 5316	MSR_RELOAD_PMCx	Core	Reload value for IA32_PMCx (R/W)
		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 CPUID signature 06_86H.				

2.8 MSRS IN THE 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 with DisplayFamily_DisplayModel of 06_1AH, 06_1EH, 06_1FH, 06_2EH, see Table 2-1. Additional MSRs specific to 06_1AH, 06_1EH, 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		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination" and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Package	Platform ID (R) See Table 2-2.
17H	23	MSR_PLATFORM_ID	Package	Model Specific Platform ID (R)

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		49:0		Reserved
		52:50		See Table 2-2.
		63:53		Reserved
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	SMI Counter (R/O)
		31:0		SMI Count (R/O) Running count of SMI events since last RESET.
		63:32		Reserved
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64Processor (R/W) See Table 2-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	BIOS Update Signature ID (RO) See Table 2-2.
C1H	193	IA32_PMC0	Thread	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Thread	Performance Counter Register See Table 2-2.
C3H	195	IA32_PMC2	Thread	Performance Counter Register See Table 2-2.
C4H	196	IA32_PMC3	Thread	Performance Counter Register See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.
		7:0		Reserved
		15:8	Package	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.
		27:16		Reserved
		28	Package	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.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29	Package	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.
		39:30		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 133.33MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	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.
		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-sate 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

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		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
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	Power Management IO Redirection in C-state (R/W) See http://biosbits.org.
		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[bit10]: 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
E7H	231	IA32_MPERF	Thread	Maximum Performance Frequency Clock Count (RW) See Table 2-2.
E8H	232	IA32_APERF	Thread	Actual Performance Frequency Clock Count (RW) See Table 2-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 2-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status
		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
186H	390	IA32_PERFEVTSEL0	Thread	See Table 2-2.
		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
187H	391	IA32_PERFEVTSEL1	Thread	See Table 2-2.
188H	392	IA32_PERFEVTSEL2	Thread	See Table 2-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 2-2.
198H	408	IA32_PERF_STATUS	Core	See Table 2-2.
		15:0		Current Performance State Value.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:16		Reserved
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		0		Reserved
		3:1		On demand Clock Modulation Duty Cycle (R/W)
		4		On demand Clock Modulation Enable (R/W)
		63:5		Reserved
19BH	411	IA32_THERM_INTERRUPT	Core	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Thread	Fast-Strings Enable See Table 2-2.
		2:1		Reserved
		3	Thread	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.
		6:4		Reserved
		7	Thread	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Thread	Branch Trace Storage Unavailable (RO) See Table 2-2.
		12	Thread	Processor Event Based Sampling Unavailable (RO) See Table 2-2.
		15:13		Reserved
		16	Package	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 2-2.
		21:19		Reserved
		22	Thread	Limit CPUID Maxval (R/W) See Table 2-2.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		23	Thread	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Thread	XD Bit Disable (R/W) See Table 2-2.
		37:35		Reserved
		38	Package	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.
		63:39		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Thread	Temperature Target
		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
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	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.
		1	Core	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).
		2	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		3	Core	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.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:4		Reserved
1A6H	422	MSR_OFFCORE_RSP_0	Thread	Offcore Response Event Select Register (R/W)
1AAH	426	MSR_MISC_PWR_MGMT		Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org.
		0	Package	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.
		1	Thread	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.(EAX=06h):ECX[3].
		63:2		Reserved
1ACH	428	MSR_TURBO_POWER_CURRENT_LIMIT		See http://biosbits.org.
		14:0	Package	TDP Limit (R/W) TDP limit in 1/8 Watt granularity.
		15	Package	TDP Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.
		30:16	Package	TDC Limit (R/W) TDC limit in 1/8 Amp granularity.
		31	Package	TDC Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.
		63:32		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1C8H	456	MSR_LBR_SELECT	Core	Last Branch Record Filtering Select Register (R/W) See Section 17.9.2, "Filtering of Last Branch Records."
		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
1C9H	457	MSR_LASTBRANCH_TOS	Thread	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).
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W) See Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Thread	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.
1DEH	478	MSR_LER_TO_LIP	Thread	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.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register See http://biosbits.org.
		0		Reserved
		1	Package	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).
		63:2		Reserved

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 2-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 2-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 2-2.
277H	631	IA32_PAT	Thread	See Table 2-2.
280H	640	IA32_MC0_CTL2	Package	See Table 2-2.
281H	641	IA32_MC1_CTL2	Package	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
284H	644	IA32_MC4_CTL2	Core	See Table 2-2.
285H	645	IA32_MC5_CTL2	Core	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Thread	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		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
38DH	909	IA32_FIXED_CTR_CTRL	Thread	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STATUS	Thread	Provides single-bit status used by software to query the overflow condition of each performance counter. (RO)
		61		UNC_Ovf Uncore overflowed if 1.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 2-2. See Section 19.6.2.2, "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.
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Thread	(R/W)
		61		CLR_UNC_Ovf Set 1 to clear UNC_Ovf.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Section 19.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		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
3F6H	1014	MSR_PEBS_LD_LAT	Thread	See Section 19.3.1.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
400H	1024	IA32_MC0_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	IA32_MC0_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV 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.
407H	1031	IA32_MC1_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV 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.
40BH	1035	IA32_MC2_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV 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.
40FH	1039	IA32_MC3_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
410H	1040	IA32_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC4_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV 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.
413H	1043	IA32_MC4_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
414H	1044	IA32_MC5_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
416H	1046	IA32_MC5_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	IA32_MC5_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
41AH	1050	IA32_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	IA32_MC6_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
41DH	1053	IA32_MC7_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
41EH	1054	IA32_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
41FH	1055	IA32_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	IA32_MC8_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
422H	1058	IA32_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	IA32_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
480H	1152	IA32_VMX_BASIC	Thread	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTLS	Thread	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTLS	Thread	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTLS	Thread	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Thread	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Thread	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Thread	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
489H	1161	IA32_VMX_CR4_FIXED1	Thread	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Thread	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Thread	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	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. ▪ Section 17.9.1 and record format in Section 17.4.8.1.
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Thread	Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Thread	Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Thread	Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Thread	Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Thread	Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Thread	Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Thread	Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Thread	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.
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Thread	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Thread	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Thread	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Thread	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Thread	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Thread	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Thread	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Thread	Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Thread	Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Thread	Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Thread	Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Thread	Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Thread	Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Thread	Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Thread	Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
802H	2050	IA32_X2APIC_APICID	Thread	x2APIC ID Register (R/O)
803H	2051	IA32_X2APIC_VERSION	Thread	x2APIC Version Register (R/O)
808H	2056	IA32_X2APIC_TPR	Thread	x2APIC Task Priority Register (R/W)
80AH	2058	IA32_X2APIC_PPR	Thread	x2APIC Processor Priority Register (R/O)
80BH	2059	IA32_X2APIC_EOI	Thread	x2APIC EOI Register (W/O)
80DH	2061	IA32_X2APIC_LDR	Thread	x2APIC Logical Destination Register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Thread	x2APIC Spurious Interrupt Vector Register (R/W)
810H	2064	IA32_X2APIC_ISR0	Thread	x2APIC In-Service Register Bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Thread	x2APIC In-Service Register Bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Thread	x2APIC In-Service Register Bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Thread	x2APIC In-Service Register Bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Thread	x2APIC In-Service Register Bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service Register Bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service Register Bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service Register Bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Thread	x2APIC Trigger Mode Register Bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode Register Bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode Register Bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode Register Bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode Register Bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode Register Bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode Register Bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Thread	x2APIC Trigger Mode Register Bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Thread	x2APIC Interrupt Request Register Bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Thread	x2APIC Interrupt Request Register Bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Thread	x2APIC Interrupt Request Register Bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Thread	x2APIC Interrupt Request Register Bits [127:96] (R/O)
824H	2084	IA32_X2APIC_IRR4	Thread	x2APIC Interrupt Request Register Bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Thread	x2APIC Interrupt Request Register Bits [191:160] (R/O)

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
826H	2086	IA32_X2APIC_IRR6	Thread	x2APIC Interrupt Request Register Bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request Register Bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status Register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt Register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command Register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt Register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt Register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor Register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 Register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 Register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error Register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count Register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count Register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration Register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI Register (W/O)
C000_0080H		IA32_EFER	Thread	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Thread	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Thread	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Thread	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	Swap Target of BASE Address of GS (R/W) See Table 2-2.
C000_0103H		IA32_TSC_AUX	Thread	AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 17.17.2, "IA32_TSC_AUX Register and RDTSCP Support."

2.8.1 Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

Intel Xeon Processor 5500 and 3400 series support additional model-specific registers listed in Table 2-16. These MSRs also apply to Intel Core i7 and i5 processor family CPUID signature with DisplayFamily_DisplayModel of 06_1AH, 06_1EH and 06_1FH, see Table 2-1.

Table 2-16. Additional MSRs in Intel® Xeon® Processor 5500 and 3400 Series

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Actual maximum turbo frequency is multiplied by 133.33MHz. (Not available in model 06_2EH.)
		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
301H	769	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)
		5		From E to I (R/W)
		6		From S to I (R/W)
		7		From F to I (R/W)
		63:8		Reserved
391H	913	MSR_UNCORE_PERF_GLOBAL_CTRL	Package	See Section 19.3.1.2.1, “Uncore Performance Monitoring Management Facility.”
392H	914	MSR_UNCORE_PERF_GLOBAL_STATUS	Package	See Section 19.3.1.2.1, “Uncore Performance Monitoring Management Facility.”
393H	915	MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	Package	See Section 19.3.1.2.1, “Uncore Performance Monitoring Management Facility.”
394H	916	MSR_UNCORE_FIXED_CTR0	Package	See Section 19.3.1.2.1, “Uncore Performance Monitoring Management Facility.”
395H	917	MSR_UNCORE_FIXED_CTR_CTRL	Package	See Section 19.3.1.2.1, “Uncore Performance Monitoring Management Facility.”
396H	918	MSR_UNCORE_ADDR_OPCODE_MATCH	Package	See Section 19.3.1.2.3, “Uncore Address/Opcode Match MSR.”
3B0H	960	MSR_UNCORE_PMC0	Package	See Section 19.3.1.2.2, “Uncore Performance Event Configuration Facility.”
3B1H	961	MSR_UNCORE_PMC1	Package	See Section 19.3.1.2.2, “Uncore Performance Event Configuration Facility.”
3B2H	962	MSR_UNCORE_PMC2	Package	See Section 19.3.1.2.2, “Uncore Performance Event Configuration Facility.”

Table 2-16. Additional MSRs in Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3B3H	963	MSR_UNCORE_PMC3	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B4H	964	MSR_UNCORE_PMC4	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B5H	965	MSR_UNCORE_PMC5	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B6H	966	MSR_UNCORE_PMC6	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3B7H	967	MSR_UNCORE_PMC7	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C0H	944	MSR_UNCORE_PERFEVTSEL0	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C1H	945	MSR_UNCORE_PERFEVTSEL1	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C2H	946	MSR_UNCORE_PERFEVTSEL2	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C3H	947	MSR_UNCORE_PERFEVTSEL3	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C4H	948	MSR_UNCORE_PERFEVTSEL4	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C5H	949	MSR_UNCORE_PERFEVTSEL5	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C6H	950	MSR_UNCORE_PERFEVTSEL6	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."
3C7H	951	MSR_UNCORE_PERFEVTSEL7	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."

2.8.2 Additional MSRs in the Intel® Xeon® Processor 7500 Series

Intel Xeon Processor 7500 series support 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 with DisplayFamily_DisplayModel of 06_2EH.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Reserved Attempt to read/write will cause #UD.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
394H	816	MSR_W_PMON_FIXED_CTR	Package	Uncore W-box perfmon fixed counter.
395H	817	MSR_W_PMON_FIXED_CTR_CTL	Package	Uncore U-box perfmon fixed counter control MSR.
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
425H	1061	IA32_MC9_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
426H	1062	IA32_MC9_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
427H	1063	IA32_MC9_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
429H	1065	IA32_MC10_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
42AH	1066	IA32_MC10_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42BH	1067	IA32_MC10_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	IA32_MC11_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
42EH	1070	IA32_MC11_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42FH	1071	IA32_MC11_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
431H	1073	IA32_MC12_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
432H	1074	IA32_MC12_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
433H	1075	IA32_MC12_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
435H	1077	IA32_MC13_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
436H	1078	IA32_MC13_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
437H	1079	IA32_MC13_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
439H	1081	IA32_MC14_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
43AH	1082	IA32_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	IA32_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
43DH	1085	IA32_MC15_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
43EH	1086	IA32_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43FH	1087	IA32_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
441H	1089	IA32_MC16_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
442H	1090	IA32_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
443H	1091	IA32_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	IA32_MC17_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
446H	1094	IA32_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	IA32_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	IA32_MC18_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
44AH	1098	IA32_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	IA32_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	IA32_MC19_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
44EH	1102	IA32_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	IA32_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
451H	1105	IA32_MC20_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
452H	1106	IA32_MC20_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
453H	1107	IA32_MC20_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
455H	1109	IA32_MC21_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
456H	1110	IA32_MC21_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
457H	1111	IA32_MC21_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
C00H	3072	MSR_U_PMON_GLOBAL_CTRL	Package	Uncore U-box perfmon global control MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C01H	3073	MSR_U_PMON_GLOBAL_STATUS	Package	Uncore U-box perfmon global status MSR.
C02H	3074	MSR_U_PMON_GLOBAL_OVF_CTRL	Package	Uncore U-box perfmon global overflow control MSR.
C10H	3088	MSR_U_PMON_EVNT_SEL	Package	Uncore U-box perfmon event select MSR.
C11H	3089	MSR_U_PMON_CTR	Package	Uncore U-box perfmon counter MSR.
C20H	3104	MSR_B0_PMON_BOX_CTRL	Package	Uncore B-box 0 perfmon local box control MSR.
C21H	3105	MSR_B0_PMON_BOX_STATUS	Package	Uncore B-box 0 perfmon local box status MSR.
C22H	3106	MSR_B0_PMON_BOX_OVF_CTRL	Package	Uncore B-box 0 perfmon local box overflow control MSR.
C30H	3120	MSR_B0_PMON_EVNT_SEL0	Package	Uncore B-box 0 perfmon event select MSR.
C31H	3121	MSR_B0_PMON_CTR0	Package	Uncore B-box 0 perfmon counter MSR.
C32H	3122	MSR_B0_PMON_EVNT_SEL1	Package	Uncore B-box 0 perfmon event select MSR.
C33H	3123	MSR_B0_PMON_CTR1	Package	Uncore B-box 0 perfmon counter MSR.
C34H	3124	MSR_B0_PMON_EVNT_SEL2	Package	Uncore B-box 0 perfmon event select MSR.
C35H	3125	MSR_B0_PMON_CTR2	Package	Uncore B-box 0 perfmon counter MSR.
C36H	3126	MSR_B0_PMON_EVNT_SEL3	Package	Uncore B-box 0 perfmon event select MSR.
C37H	3127	MSR_B0_PMON_CTR3	Package	Uncore B-box 0 perfmon counter MSR.
C40H	3136	MSR_S0_PMON_BOX_CTRL	Package	Uncore S-box 0 perfmon local box control MSR.
C41H	3137	MSR_S0_PMON_BOX_STATUS	Package	Uncore S-box 0 perfmon local box status MSR.
C42H	3138	MSR_S0_PMON_BOX_OVF_CTRL	Package	Uncore S-box 0 perfmon local box overflow control MSR.
C50H	3152	MSR_S0_PMON_EVNT_SEL0	Package	Uncore S-box 0 perfmon event select MSR.
C51H	3153	MSR_S0_PMON_CTR0	Package	Uncore S-box 0 perfmon counter MSR.
C52H	3154	MSR_S0_PMON_EVNT_SEL1	Package	Uncore S-box 0 perfmon event select MSR.
C53H	3155	MSR_S0_PMON_CTR1	Package	Uncore S-box 0 perfmon counter MSR.
C54H	3156	MSR_S0_PMON_EVNT_SEL2	Package	Uncore S-box 0 perfmon event select MSR.
C55H	3157	MSR_S0_PMON_CTR2	Package	Uncore S-box 0 perfmon counter MSR.
C56H	3158	MSR_S0_PMON_EVNT_SEL3	Package	Uncore S-box 0 perfmon event select MSR.
C57H	3159	MSR_S0_PMON_CTR3	Package	Uncore S-box 0 perfmon counter MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C60H	3168	MSR_B1_PMON_BOX_CTRL	Package	Uncore B-box 1 perfmon local box control MSR.
C61H	3169	MSR_B1_PMON_BOX_STATUS	Package	Uncore B-box 1 perfmon local box status MSR.
C62H	3170	MSR_B1_PMON_BOX_OVF_CTRL	Package	Uncore B-box 1 perfmon local box overflow control MSR.
C70H	3184	MSR_B1_PMON_EVNT_SEL0	Package	Uncore B-box 1 perfmon event select MSR.
C71H	3185	MSR_B1_PMON_CTR0	Package	Uncore B-box 1 perfmon counter MSR.
C72H	3186	MSR_B1_PMON_EVNT_SEL1	Package	Uncore B-box 1 perfmon event select MSR.
C73H	3187	MSR_B1_PMON_CTR1	Package	Uncore B-box 1 perfmon counter MSR.
C74H	3188	MSR_B1_PMON_EVNT_SEL2	Package	Uncore B-box 1 perfmon event select MSR.
C75H	3189	MSR_B1_PMON_CTR2	Package	Uncore B-box 1 perfmon counter MSR.
C76H	3190	MSR_B1_PMON_EVNT_SEL3	Package	Uncore B-box 1vperfmon event select MSR.
C77H	3191	MSR_B1_PMON_CTR3	Package	Uncore B-box 1 perfmon counter MSR.
C80H	3120	MSR_W_PMON_BOX_CTRL	Package	Uncore W-box perfmon local box control MSR.
C81H	3121	MSR_W_PMON_BOX_STATUS	Package	Uncore W-box perfmon local box status MSR.
C82H	3122	MSR_W_PMON_BOX_OVF_CTRL	Package	Uncore W-box perfmon local box overflow control MSR.
C90H	3136	MSR_W_PMON_EVNT_SEL0	Package	Uncore W-box perfmon event select MSR.
C91H	3137	MSR_W_PMON_CTR0	Package	Uncore W-box perfmon counter MSR.
C92H	3138	MSR_W_PMON_EVNT_SEL1	Package	Uncore W-box perfmon event select MSR.
C93H	3139	MSR_W_PMON_CTR1	Package	Uncore W-box perfmon counter MSR.
C94H	3140	MSR_W_PMON_EVNT_SEL2	Package	Uncore W-box perfmon event select MSR.
C95H	3141	MSR_W_PMON_CTR2	Package	Uncore W-box perfmon counter MSR.
C96H	3142	MSR_W_PMON_EVNT_SEL3	Package	Uncore W-box perfmon event select MSR.
C97H	3143	MSR_W_PMON_CTR3	Package	Uncore W-box perfmon counter MSR.
CA0H	3232	MSR_M0_PMON_BOX_CTRL	Package	Uncore M-box 0 perfmon local box control MSR.
CA1H	3233	MSR_M0_PMON_BOX_STATUS	Package	Uncore M-box 0 perfmon local box status MSR.
CA2H	3234	MSR_M0_PMON_BOX_OVF_CTRL	Package	Uncore M-box 0 perfmon local box overflow control MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CA4H	3236	MSR_M0_PMON_TIMESTAMP	Package	Uncore M-box 0 perfmon time stamp unit select MSR.
CA5H	3237	MSR_M0_PMON_DSP	Package	Uncore M-box 0 perfmon DSP unit select MSR.
CA6H	3238	MSR_M0_PMON_ISS	Package	Uncore M-box 0 perfmon ISS unit select MSR.
CA7H	3239	MSR_M0_PMON_MAP	Package	Uncore M-box 0 perfmon MAP unit select MSR.
CA8H	3240	MSR_M0_PMON_MSC_THR	Package	Uncore M-box 0 perfmon MIC THR select MSR.
CA9H	3241	MSR_M0_PMON_PGT	Package	Uncore M-box 0 perfmon PGT unit select MSR.
CAAH	3242	MSR_M0_PMON_PLD	Package	Uncore M-box 0 perfmon PLD unit select MSR.
CABH	3243	MSR_M0_PMON_ZDP	Package	Uncore M-box 0 perfmon ZDP unit select MSR.
CB0H	3248	MSR_M0_PMON_EVNT_SEL0	Package	Uncore M-box 0 perfmon event select MSR.
CB1H	3249	MSR_M0_PMON_CTR0	Package	Uncore M-box 0 perfmon counter MSR.
CB2H	3250	MSR_M0_PMON_EVNT_SEL1	Package	Uncore M-box 0 perfmon event select MSR.
CB3H	3251	MSR_M0_PMON_CTR1	Package	Uncore M-box 0 perfmon counter MSR.
CB4H	3252	MSR_M0_PMON_EVNT_SEL2	Package	Uncore M-box 0 perfmon event select MSR.
CB5H	3253	MSR_M0_PMON_CTR2	Package	Uncore M-box 0 perfmon counter MSR.
CB6H	3254	MSR_M0_PMON_EVNT_SEL3	Package	Uncore M-box 0 perfmon event select MSR.
CB7H	3255	MSR_M0_PMON_CTR3	Package	Uncore M-box 0 perfmon counter MSR.
CB8H	3256	MSR_M0_PMON_EVNT_SEL4	Package	Uncore M-box 0 perfmon event select MSR.
CB9H	3257	MSR_M0_PMON_CTR4	Package	Uncore M-box 0 perfmon counter MSR.
CBAH	3258	MSR_M0_PMON_EVNT_SEL5	Package	Uncore M-box 0 perfmon event select MSR.
CBBH	3259	MSR_M0_PMON_CTR5	Package	Uncore M-box 0 perfmon counter MSR.
CC0H	3264	MSR_S1_PMON_BOX_CTRL	Package	Uncore S-box 1 perfmon local box control MSR.
CC1H	3265	MSR_S1_PMON_BOX_STATUS	Package	Uncore S-box 1 perfmon local box status MSR.
CC2H	3266	MSR_S1_PMON_BOX_OVF_CTRL	Package	Uncore S-box 1 perfmon local box overflow control MSR.
CD0H	3280	MSR_S1_PMON_EVNT_SEL0	Package	Uncore S-box 1 perfmon event select MSR.
CD1H	3281	MSR_S1_PMON_CTR0	Package	Uncore S-box 1 perfmon counter MSR.
CD2H	3282	MSR_S1_PMON_EVNT_SEL1	Package	Uncore S-box 1 perfmon event select MSR.
CD3H	3283	MSR_S1_PMON_CTR1	Package	Uncore S-box 1 perfmon counter MSR.
CD4H	3284	MSR_S1_PMON_EVNT_SEL2	Package	Uncore S-box 1 perfmon event select MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CD5H	3285	MSR_S1_PMON_CTR2	Package	Uncore S-box 1 perfmon counter MSR.
CD6H	3286	MSR_S1_PMON_EVNT_SEL3	Package	Uncore S-box 1 perfmon event select MSR.
CD7H	3287	MSR_S1_PMON_CTR3	Package	Uncore S-box 1 perfmon counter MSR.
CE0H	3296	MSR_M1_PMON_BOX_CTRL	Package	Uncore M-box 1 perfmon local box control MSR.
CE1H	3297	MSR_M1_PMON_BOX_STATUS	Package	Uncore M-box 1 perfmon local box status MSR.
CE2H	3298	MSR_M1_PMON_BOX_OVF_CTRL	Package	Uncore M-box 1 perfmon local box overflow control MSR.
CE4H	3300	MSR_M1_PMON_TIMESTAMP	Package	Uncore M-box 1 perfmon time stamp unit select MSR.
CE5H	3301	MSR_M1_PMON_DSP	Package	Uncore M-box 1 perfmon DSP unit select MSR.
CE6H	3302	MSR_M1_PMON_ISS	Package	Uncore M-box 1 perfmon ISS unit select MSR.
CE7H	3303	MSR_M1_PMON_MAP	Package	Uncore M-box 1 perfmon MAP unit select MSR.
CE8H	3304	MSR_M1_PMON_MSC_THR	Package	Uncore M-box 1 perfmon MIC THR select MSR.
CE9H	3305	MSR_M1_PMON_PGT	Package	Uncore M-box 1 perfmon PGT unit select MSR.
CEAH	3306	MSR_M1_PMON_PLD	Package	Uncore M-box 1 perfmon PLD unit select MSR.
CEBH	3307	MSR_M1_PMON_ZDP	Package	Uncore M-box 1 perfmon ZDP unit select MSR.
CF0H	3312	MSR_M1_PMON_EVNT_SEL0	Package	Uncore M-box 1 perfmon event select MSR.
CF1H	3313	MSR_M1_PMON_CTR0	Package	Uncore M-box 1 perfmon counter MSR.
CF2H	3314	MSR_M1_PMON_EVNT_SEL1	Package	Uncore M-box 1 perfmon event select MSR.
CF3H	3315	MSR_M1_PMON_CTR1	Package	Uncore M-box 1 perfmon counter MSR.
CF4H	3316	MSR_M1_PMON_EVNT_SEL2	Package	Uncore M-box 1 perfmon event select MSR.
CF5H	3317	MSR_M1_PMON_CTR2	Package	Uncore M-box 1 perfmon counter MSR.
CF6H	3318	MSR_M1_PMON_EVNT_SEL3	Package	Uncore M-box 1 perfmon event select MSR.
CF7H	3319	MSR_M1_PMON_CTR3	Package	Uncore M-box 1 perfmon counter MSR.
CF8H	3320	MSR_M1_PMON_EVNT_SEL4	Package	Uncore M-box 1 perfmon event select MSR.
CF9H	3321	MSR_M1_PMON_CTR4	Package	Uncore M-box 1 perfmon counter MSR.
CFAH	3322	MSR_M1_PMON_EVNT_SEL5	Package	Uncore M-box 1 perfmon event select MSR.
CFBH	3323	MSR_M1_PMON_CTR5	Package	Uncore M-box 1 perfmon counter MSR.
D00H	3328	MSR_C0_PMON_BOX_CTRL	Package	Uncore C-box 0 perfmon local box control MSR.
D01H	3329	MSR_C0_PMON_BOX_STATUS	Package	Uncore C-box 0 perfmon local box status MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D02H	3330	MSR_C0_PMON_BOX_OVF_CTRL	Package	Uncore C-box 0 perfmon local box overflow control MSR.
D10H	3344	MSR_C0_PMON_EVNT_SEL0	Package	Uncore C-box 0 perfmon event select MSR.
D11H	3345	MSR_C0_PMON_CTR0	Package	Uncore C-box 0 perfmon counter MSR.
D12H	3346	MSR_C0_PMON_EVNT_SEL1	Package	Uncore C-box 0 perfmon event select MSR.
D13H	3347	MSR_C0_PMON_CTR1	Package	Uncore C-box 0 perfmon counter MSR.
D14H	3348	MSR_C0_PMON_EVNT_SEL2	Package	Uncore C-box 0 perfmon event select MSR.
D15H	3349	MSR_C0_PMON_CTR2	Package	Uncore C-box 0 perfmon counter MSR.
D16H	3350	MSR_C0_PMON_EVNT_SEL3	Package	Uncore C-box 0 perfmon event select MSR.
D17H	3351	MSR_C0_PMON_CTR3	Package	Uncore C-box 0 perfmon counter MSR.
D18H	3352	MSR_C0_PMON_EVNT_SEL4	Package	Uncore C-box 0 perfmon event select MSR.
D19H	3353	MSR_C0_PMON_CTR4	Package	Uncore C-box 0 perfmon counter MSR.
D1AH	3354	MSR_C0_PMON_EVNT_SEL5	Package	Uncore C-box 0 perfmon event select MSR.
D1BH	3355	MSR_C0_PMON_CTR5	Package	Uncore C-box 0 perfmon counter MSR.
D20H	3360	MSR_C4_PMON_BOX_CTRL	Package	Uncore C-box 4 perfmon local box control MSR.
D21H	3361	MSR_C4_PMON_BOX_STATUS	Package	Uncore C-box 4 perfmon local box status MSR.
D22H	3362	MSR_C4_PMON_BOX_OVF_CTRL	Package	Uncore C-box 4 perfmon local box overflow control MSR.
D30H	3376	MSR_C4_PMON_EVNT_SEL0	Package	Uncore C-box 4 perfmon event select MSR.
D31H	3377	MSR_C4_PMON_CTR0	Package	Uncore C-box 4 perfmon counter MSR.
D32H	3378	MSR_C4_PMON_EVNT_SEL1	Package	Uncore C-box 4 perfmon event select MSR.
D33H	3379	MSR_C4_PMON_CTR1	Package	Uncore C-box 4 perfmon counter MSR.
D34H	3380	MSR_C4_PMON_EVNT_SEL2	Package	Uncore C-box 4 perfmon event select MSR.
D35H	3381	MSR_C4_PMON_CTR2	Package	Uncore C-box 4 perfmon counter MSR.
D36H	3382	MSR_C4_PMON_EVNT_SEL3	Package	Uncore C-box 4 perfmon event select MSR.
D37H	3383	MSR_C4_PMON_CTR3	Package	Uncore C-box 4 perfmon counter MSR.
D38H	3384	MSR_C4_PMON_EVNT_SEL4	Package	Uncore C-box 4 perfmon event select MSR.
D39H	3385	MSR_C4_PMON_CTR4	Package	Uncore C-box 4 perfmon counter MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D3AH	3386	MSR_C4_PMON_EVNT_SEL5	Package	Uncore C-box 4 perfmon event select MSR.
D3BH	3387	MSR_C4_PMON_CTR5	Package	Uncore C-box 4 perfmon counter MSR.
D40H	3392	MSR_C2_PMON_BOX_CTRL	Package	Uncore C-box 2 perfmon local box control MSR.
D41H	3393	MSR_C2_PMON_BOX_STATUS	Package	Uncore C-box 2 perfmon local box status MSR.
D42H	3394	MSR_C2_PMON_BOX_OVF_CTRL	Package	Uncore C-box 2 perfmon local box overflow control MSR.
D50H	3408	MSR_C2_PMON_EVNT_SEL0	Package	Uncore C-box 2 perfmon event select MSR.
D51H	3409	MSR_C2_PMON_CTR0	Package	Uncore C-box 2 perfmon counter MSR.
D52H	3410	MSR_C2_PMON_EVNT_SEL1	Package	Uncore C-box 2 perfmon event select MSR.
D53H	3411	MSR_C2_PMON_CTR1	Package	Uncore C-box 2 perfmon counter MSR.
D54H	3412	MSR_C2_PMON_EVNT_SEL2	Package	Uncore C-box 2 perfmon event select MSR.
D55H	3413	MSR_C2_PMON_CTR2	Package	Uncore C-box 2 perfmon counter MSR.
D56H	3414	MSR_C2_PMON_EVNT_SEL3	Package	Uncore C-box 2 perfmon event select MSR.
D57H	3415	MSR_C2_PMON_CTR3	Package	Uncore C-box 2 perfmon counter MSR.
D58H	3416	MSR_C2_PMON_EVNT_SEL4	Package	Uncore C-box 2 perfmon event select MSR.
D59H	3417	MSR_C2_PMON_CTR4	Package	Uncore C-box 2 perfmon counter MSR.
D5AH	3418	MSR_C2_PMON_EVNT_SEL5	Package	Uncore C-box 2 perfmon event select MSR.
D5BH	3419	MSR_C2_PMON_CTR5	Package	Uncore C-box 2 perfmon counter MSR.
D60H	3424	MSR_C6_PMON_BOX_CTRL	Package	Uncore C-box 6 perfmon local box control MSR.
D61H	3425	MSR_C6_PMON_BOX_STATUS	Package	Uncore C-box 6 perfmon local box status MSR.
D62H	3426	MSR_C6_PMON_BOX_OVF_CTRL	Package	Uncore C-box 6 perfmon local box overflow control MSR.
D70H	3440	MSR_C6_PMON_EVNT_SEL0	Package	Uncore C-box 6 perfmon event select MSR.
D71H	3441	MSR_C6_PMON_CTR0	Package	Uncore C-box 6 perfmon counter MSR.
D72H	3442	MSR_C6_PMON_EVNT_SEL1	Package	Uncore C-box 6 perfmon event select MSR.
D73H	3443	MSR_C6_PMON_CTR1	Package	Uncore C-box 6 perfmon counter MSR.
D74H	3444	MSR_C6_PMON_EVNT_SEL2	Package	Uncore C-box 6 perfmon event select MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D75H	3445	MSR_C6_PMON_CTR2	Package	Uncore C-box 6 perfmon counter MSR.
D76H	3446	MSR_C6_PMON_EVNT_SEL3	Package	Uncore C-box 6 perfmon event select MSR.
D77H	3447	MSR_C6_PMON_CTR3	Package	Uncore C-box 6 perfmon counter MSR.
D78H	3448	MSR_C6_PMON_EVNT_SEL4	Package	Uncore C-box 6 perfmon event select MSR.
D79H	3449	MSR_C6_PMON_CTR4	Package	Uncore C-box 6 perfmon counter MSR.
D7AH	3450	MSR_C6_PMON_EVNT_SEL5	Package	Uncore C-box 6 perfmon event select MSR.
D7BH	3451	MSR_C6_PMON_CTR5	Package	Uncore C-box 6 perfmon counter MSR.
D80H	3456	MSR_C1_PMON_BOX_CTRL	Package	Uncore C-box 1 perfmon local box control MSR.
D81H	3457	MSR_C1_PMON_BOX_STATUS	Package	Uncore C-box 1 perfmon local box status MSR.
D82H	3458	MSR_C1_PMON_BOX_OVF_CTRL	Package	Uncore C-box 1 perfmon local box overflow control MSR.
D90H	3472	MSR_C1_PMON_EVNT_SEL0	Package	Uncore C-box 1 perfmon event select MSR.
D91H	3473	MSR_C1_PMON_CTR0	Package	Uncore C-box 1 perfmon counter MSR.
D92H	3474	MSR_C1_PMON_EVNT_SEL1	Package	Uncore C-box 1 perfmon event select MSR.
D93H	3475	MSR_C1_PMON_CTR1	Package	Uncore C-box 1 perfmon counter MSR.
D94H	3476	MSR_C1_PMON_EVNT_SEL2	Package	Uncore C-box 1 perfmon event select MSR.
D95H	3477	MSR_C1_PMON_CTR2	Package	Uncore C-box 1 perfmon counter MSR.
D96H	3478	MSR_C1_PMON_EVNT_SEL3	Package	Uncore C-box 1 perfmon event select MSR.
D97H	3479	MSR_C1_PMON_CTR3	Package	Uncore C-box 1 perfmon counter MSR.
D98H	3480	MSR_C1_PMON_EVNT_SEL4	Package	Uncore C-box 1 perfmon event select MSR.
D99H	3481	MSR_C1_PMON_CTR4	Package	Uncore C-box 1 perfmon counter MSR.
D9AH	3482	MSR_C1_PMON_EVNT_SEL5	Package	Uncore C-box 1 perfmon event select MSR.
D9BH	3483	MSR_C1_PMON_CTR5	Package	Uncore C-box 1 perfmon counter MSR.
DA0H	3488	MSR_C5_PMON_BOX_CTRL	Package	Uncore C-box 5 perfmon local box control MSR.
DA1H	3489	MSR_C5_PMON_BOX_STATUS	Package	Uncore C-box 5 perfmon local box status MSR.
DA2H	3490	MSR_C5_PMON_BOX_OVF_CTRL	Package	Uncore C-box 5 perfmon local box overflow control MSR.
DB0H	3504	MSR_C5_PMON_EVNT_SEL0	Package	Uncore C-box 5 perfmon event select MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DB1H	3505	MSR_C5_PMON_CTR0	Package	Uncore C-box 5 perfmon counter MSR.
DB2H	3506	MSR_C5_PMON_EVNT_SEL1	Package	Uncore C-box 5 perfmon event select MSR.
DB3H	3507	MSR_C5_PMON_CTR1	Package	Uncore C-box 5 perfmon counter MSR.
DB4H	3508	MSR_C5_PMON_EVNT_SEL2	Package	Uncore C-box 5 perfmon event select MSR.
DB5H	3509	MSR_C5_PMON_CTR2	Package	Uncore C-box 5 perfmon counter MSR.
DB6H	3510	MSR_C5_PMON_EVNT_SEL3	Package	Uncore C-box 5 perfmon event select MSR.
DB7H	3511	MSR_C5_PMON_CTR3	Package	Uncore C-box 5 perfmon counter MSR.
DB8H	3512	MSR_C5_PMON_EVNT_SEL4	Package	Uncore C-box 5 perfmon event select MSR.
DB9H	3513	MSR_C5_PMON_CTR4	Package	Uncore C-box 5 perfmon counter MSR.
DBAH	3514	MSR_C5_PMON_EVNT_SEL5	Package	Uncore C-box 5 perfmon event select MSR.
DBBH	3515	MSR_C5_PMON_CTR5	Package	Uncore C-box 5 perfmon counter MSR.
DC0H	3520	MSR_C3_PMON_BOX_CTRL	Package	Uncore C-box 3 perfmon local box control MSR.
DC1H	3521	MSR_C3_PMON_BOX_STATUS	Package	Uncore C-box 3 perfmon local box status MSR.
DC2H	3522	MSR_C3_PMON_BOX_OVF_CTRL	Package	Uncore C-box 3 perfmon local box overflow control MSR.
DD0H	3536	MSR_C3_PMON_EVNT_SEL0	Package	Uncore C-box 3 perfmon event select MSR.
DD1H	3537	MSR_C3_PMON_CTR0	Package	Uncore C-box 3 perfmon counter MSR.
DD2H	3538	MSR_C3_PMON_EVNT_SEL1	Package	Uncore C-box 3 perfmon event select MSR.
DD3H	3539	MSR_C3_PMON_CTR1	Package	Uncore C-box 3 perfmon counter MSR.
DD4H	3540	MSR_C3_PMON_EVNT_SEL2	Package	Uncore C-box 3 perfmon event select MSR.
DD5H	3541	MSR_C3_PMON_CTR2	Package	Uncore C-box 3 perfmon counter MSR.
DD6H	3542	MSR_C3_PMON_EVNT_SEL3	Package	Uncore C-box 3 perfmon event select MSR.
DD7H	3543	MSR_C3_PMON_CTR3	Package	Uncore C-box 3 perfmon counter MSR.
DD8H	3544	MSR_C3_PMON_EVNT_SEL4	Package	Uncore C-box 3 perfmon event select MSR.
DD9H	3545	MSR_C3_PMON_CTR4	Package	Uncore C-box 3 perfmon counter MSR.
DDAH	3546	MSR_C3_PMON_EVNT_SEL5	Package	Uncore C-box 3 perfmon event select MSR.
DDBH	3547	MSR_C3_PMON_CTR5	Package	Uncore C-box 3 perfmon counter MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DE0H	3552	MSR_C7_PMON_BOX_CTRL	Package	Uncore C-box 7 perfmon local box control MSR.
DE1H	3553	MSR_C7_PMON_BOX_STATUS	Package	Uncore C-box 7 perfmon local box status MSR.
DE2H	3554	MSR_C7_PMON_BOX_OVF_CTRL	Package	Uncore C-box 7 perfmon local box overflow control MSR.
DF0H	3568	MSR_C7_PMON_EVNT_SEL0	Package	Uncore C-box 7 perfmon event select MSR.
DF1H	3569	MSR_C7_PMON_CTR0	Package	Uncore C-box 7 perfmon counter MSR.
DF2H	3570	MSR_C7_PMON_EVNT_SEL1	Package	Uncore C-box 7 perfmon event select MSR.
DF3H	3571	MSR_C7_PMON_CTR1	Package	Uncore C-box 7 perfmon counter MSR.
DF4H	3572	MSR_C7_PMON_EVNT_SEL2	Package	Uncore C-box 7 perfmon event select MSR.
DF5H	3573	MSR_C7_PMON_CTR2	Package	Uncore C-box 7 perfmon counter MSR.
DF6H	3574	MSR_C7_PMON_EVNT_SEL3	Package	Uncore C-box 7 perfmon event select MSR.
DF7H	3575	MSR_C7_PMON_CTR3	Package	Uncore C-box 7 perfmon counter MSR.
DF8H	3576	MSR_C7_PMON_EVNT_SEL4	Package	Uncore C-box 7 perfmon event select MSR.
DF9H	3577	MSR_C7_PMON_CTR4	Package	Uncore C-box 7 perfmon counter MSR.
DFAH	3578	MSR_C7_PMON_EVNT_SEL5	Package	Uncore C-box 7 perfmon event select MSR.
DFBH	3579	MSR_C7_PMON_CTR5	Package	Uncore C-box 7 perfmon counter MSR.
E00H	3584	MSR_R0_PMON_BOX_CTRL	Package	Uncore R-box 0 perfmon local box control MSR.
E01H	3585	MSR_R0_PMON_BOX_STATUS	Package	Uncore R-box 0 perfmon local box status MSR.
E02H	3586	MSR_R0_PMON_BOX_OVF_CTRL	Package	Uncore R-box 0 perfmon local box overflow control MSR.
E04H	3588	MSR_R0_PMON_IPERF0_P0	Package	Uncore R-box 0 perfmon IPERF0 unit Port 0 select MSR.
E05H	3589	MSR_R0_PMON_IPERF0_P1	Package	Uncore R-box 0 perfmon IPERF0 unit Port 1 select MSR.
E06H	3590	MSR_R0_PMON_IPERF0_P2	Package	Uncore R-box 0 perfmon IPERF0 unit Port 2 select MSR.
E07H	3591	MSR_R0_PMON_IPERF0_P3	Package	Uncore R-box 0 perfmon IPERF0 unit Port 3 select MSR.
E08H	3592	MSR_R0_PMON_IPERF0_P4	Package	Uncore R-box 0 perfmon IPERF0 unit Port 4 select MSR.
E09H	3593	MSR_R0_PMON_IPERF0_P5	Package	Uncore R-box 0 perfmon IPERF0 unit Port 5 select MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E0AH	3594	MSR_R0_PMON_IPERF0_P6	Package	Uncore R-box 0 perfmon IPERF0 unit Port 6 select MSR.
E0BH	3595	MSR_R0_PMON_IPERF0_P7	Package	Uncore R-box 0 perfmon IPERF0 unit Port 7 select MSR.
E0CH	3596	MSR_R0_PMON_QLX_P0	Package	Uncore R-box 0 perfmon QLX unit Port 0 select MSR.
E0DH	3597	MSR_R0_PMON_QLX_P1	Package	Uncore R-box 0 perfmon QLX unit Port 1 select MSR.
E0EH	3598	MSR_R0_PMON_QLX_P2	Package	Uncore R-box 0 perfmon QLX unit Port 2 select MSR.
E0FH	3599	MSR_R0_PMON_QLX_P3	Package	Uncore R-box 0 perfmon QLX unit Port 3 select MSR.
E10H	3600	MSR_R0_PMON_EVNT_SEL0	Package	Uncore R-box 0 perfmon event select MSR.
E11H	3601	MSR_R0_PMON_CTR0	Package	Uncore R-box 0 perfmon counter MSR.
E12H	3602	MSR_R0_PMON_EVNT_SEL1	Package	Uncore R-box 0 perfmon event select MSR.
E13H	3603	MSR_R0_PMON_CTR1	Package	Uncore R-box 0 perfmon counter MSR.
E14H	3604	MSR_R0_PMON_EVNT_SEL2	Package	Uncore R-box 0 perfmon event select MSR.
E15H	3605	MSR_R0_PMON_CTR2	Package	Uncore R-box 0 perfmon counter MSR.
E16H	3606	MSR_R0_PMON_EVNT_SEL3	Package	Uncore R-box 0 perfmon event select MSR.
E17H	3607	MSR_R0_PMON_CTR3	Package	Uncore R-box 0 perfmon counter MSR.
E18H	3608	MSR_R0_PMON_EVNT_SEL4	Package	Uncore R-box 0 perfmon event select MSR.
E19H	3609	MSR_R0_PMON_CTR4	Package	Uncore R-box 0 perfmon counter MSR.
E1AH	3610	MSR_R0_PMON_EVNT_SEL5	Package	Uncore R-box 0 perfmon event select MSR.
E1BH	3611	MSR_R0_PMON_CTR5	Package	Uncore R-box 0 perfmon counter MSR.
E1CH	3612	MSR_R0_PMON_EVNT_SEL6	Package	Uncore R-box 0 perfmon event select MSR.
E1DH	3613	MSR_R0_PMON_CTR6	Package	Uncore R-box 0 perfmon counter MSR.
E1EH	3614	MSR_R0_PMON_EVNT_SEL7	Package	Uncore R-box 0 perfmon event select MSR.
E1FH	3615	MSR_R0_PMON_CTR7	Package	Uncore R-box 0 perfmon counter MSR.
E20H	3616	MSR_R1_PMON_BOX_CTRL	Package	Uncore R-box 1 perfmon local box control MSR.
E21H	3617	MSR_R1_PMON_BOX_STATUS	Package	Uncore R-box 1 perfmon local box status MSR.
E22H	3618	MSR_R1_PMON_BOX_OVF_CTRL	Package	Uncore R-box 1 perfmon local box overflow control MSR.
E24H	3620	MSR_R1_PMON_IPERF1_P8	Package	Uncore R-box 1 perfmon IPERF1 unit Port 8 select MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E25H	3621	MSR_R1_PMON_IPERF1_P9	Package	Uncore R-box 1 perfmon IPERF1 unit Port 9 select MSR.
E26H	3622	MSR_R1_PMON_IPERF1_P10	Package	Uncore R-box 1 perfmon IPERF1 unit Port 10 select MSR.
E27H	3623	MSR_R1_PMON_IPERF1_P11	Package	Uncore R-box 1 perfmon IPERF1 unit Port 11 select MSR.
E28H	3624	MSR_R1_PMON_IPERF1_P12	Package	Uncore R-box 1 perfmon IPERF1 unit Port 12 select MSR.
E29H	3625	MSR_R1_PMON_IPERF1_P13	Package	Uncore R-box 1 perfmon IPERF1 unit Port 13 select MSR.
E2AH	3626	MSR_R1_PMON_IPERF1_P14	Package	Uncore R-box 1 perfmon IPERF1 unit Port 14 select MSR.
E2BH	3627	MSR_R1_PMON_IPERF1_P15	Package	Uncore R-box 1 perfmon IPERF1 unit Port 15 select MSR.
E2CH	3628	MSR_R1_PMON_QLX_P4	Package	Uncore R-box 1 perfmon QLX unit Port 4 select MSR.
E2DH	3629	MSR_R1_PMON_QLX_P5	Package	Uncore R-box 1 perfmon QLX unit Port 5 select MSR.
E2EH	3630	MSR_R1_PMON_QLX_P6	Package	Uncore R-box 1 perfmon QLX unit Port 6 select MSR.
E2FH	3631	MSR_R1_PMON_QLX_P7	Package	Uncore R-box 1 perfmon QLX unit Port 7 select MSR.
E30H	3632	MSR_R1_PMON_EVNT_SEL8	Package	Uncore R-box 1 perfmon event select MSR.
E31H	3633	MSR_R1_PMON_CTR8	Package	Uncore R-box 1 perfmon counter MSR.
E32H	3634	MSR_R1_PMON_EVNT_SEL9	Package	Uncore R-box 1 perfmon event select MSR.
E33H	3635	MSR_R1_PMON_CTR9	Package	Uncore R-box 1 perfmon counter MSR.
E34H	3636	MSR_R1_PMON_EVNT_SEL10	Package	Uncore R-box 1 perfmon event select MSR.
E35H	3637	MSR_R1_PMON_CTR10	Package	Uncore R-box 1 perfmon counter MSR.
E36H	3638	MSR_R1_PMON_EVNT_SEL11	Package	Uncore R-box 1 perfmon event select MSR.
E37H	3639	MSR_R1_PMON_CTR11	Package	Uncore R-box 1 perfmon counter MSR.
E38H	3640	MSR_R1_PMON_EVNT_SEL12	Package	Uncore R-box 1 perfmon event select MSR.
E39H	3641	MSR_R1_PMON_CTR12	Package	Uncore R-box 1 perfmon counter MSR.
E3AH	3642	MSR_R1_PMON_EVNT_SEL13	Package	Uncore R-box 1 perfmon event select MSR.
E3BH	3643	MSR_R1_PMON_CTR13	Package	Uncore R-box 1perfmon counter MSR.
E3CH	3644	MSR_R1_PMON_EVNT_SEL14	Package	Uncore R-box 1 perfmon event select MSR.
E3DH	3645	MSR_R1_PMON_CTR14	Package	Uncore R-box 1 perfmon counter MSR.

Table 2-17. Additional MSRs in Intel® Xeon® Processor 7500 Series (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E3EH	3646	MSR_R1_PMON_EVNT_SEL15	Package	Uncore R-box 1 perfmon event select MSR.
E3FH	3647	MSR_R1_PMON_CTR15	Package	Uncore R-box 1 perfmon counter MSR.
E45H	3653	MSR_B0_PMON_MATCH	Package	Uncore B-box 0 perfmon local box match MSR.
E46H	3654	MSR_B0_PMON_MASK	Package	Uncore B-box 0 perfmon local box mask MSR.
E49H	3657	MSR_S0_PMON_MATCH	Package	Uncore S-box 0 perfmon local box match MSR.
E4AH	3658	MSR_S0_PMON_MASK	Package	Uncore S-box 0 perfmon local box mask MSR.
E4DH	3661	MSR_B1_PMON_MATCH	Package	Uncore B-box 1 perfmon local box match MSR.
E4EH	3662	MSR_B1_PMON_MASK	Package	Uncore B-box 1 perfmon local box mask MSR.
E54H	3668	MSR_M0_PMON_MM_CONFIG	Package	Uncore M-box 0 perfmon local box address match/mask config MSR.
E55H	3669	MSR_M0_PMON_ADDR_MATCH	Package	Uncore M-box 0 perfmon local box address match MSR.
E56H	3670	MSR_M0_PMON_ADDR_MASK	Package	Uncore M-box 0 perfmon local box address mask MSR.
E59H	3673	MSR_S1_PMON_MATCH	Package	Uncore S-box 1 perfmon local box match MSR.
E5AH	3674	MSR_S1_PMON_MASK	Package	Uncore S-box 1 perfmon local box mask MSR.
E5CH	3676	MSR_M1_PMON_MM_CONFIG	Package	Uncore M-box 1 perfmon local box address match/mask config MSR.
E5DH	3677	MSR_M1_PMON_ADDR_MATCH	Package	Uncore M-box 1 perfmon local box address match MSR.
E5EH	3678	MSR_M1_PMON_ADDR_MASK	Package	Uncore M-box 1 perfmon local box address mask MSR.
3B5H	965	MSR_UNCORE_PMC5	Package	See Section 19.3.1.2.2, "Uncore Performance Event Configuration Facility."

2.9 MSRS IN THE INTEL® XEON® PROCESSOR 5600 SERIES (BASED ON WESTMERE MICROARCHITECTURE)

The Intel® Xeon® Processor 5600 Series (based on Westmere microarchitecture) supports the MSR interfaces listed in Table 2-15, Table 2-16, plus additional MSR listed in Table 2-18. These MSRs apply to Intel Core i7, i5 and i3 processor family with CPUID signature DisplayFamily_DisplayModel of 06_25H and 06_2CH, see Table 2-1.

Table 2-18. Additional MSRs Supported by Intel Processors Based on Westmere Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.

Table 2-18. Additional MSRs Supported by Intel Processors Based on Westmere Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
1A7H	423	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.
		63:48		Reserved
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.

2.10 MSRS IN THE INTEL® XEON® PROCESSOR E7 FAMILY (BASED ON WESTMERE MICROARCHITECTURE)

The Intel® Xeon® Processor E7 Family (based on Westmere microarchitecture) 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 with DisplayFamily_DisplayModel of 06_2FH.

Table 2-19. Additional MSRs Supported by Intel® Xeon® Processor E7 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		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
1A7H	423	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Reserved Attempt to read/write will cause #UD.
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.
F40H	3904	MSR_C8_PMON_BOX_CTRL	Package	Uncore C-box 8 perfmon local box control MSR.
F41H	3905	MSR_C8_PMON_BOX_STATUS	Package	Uncore C-box 8 perfmon local box status MSR.
F42H	3906	MSR_C8_PMON_BOX_OVF_CTRL	Package	Uncore C-box 8 perfmon local box overflow control MSR.
F50H	3920	MSR_C8_PMON_EVNT_SEL0	Package	Uncore C-box 8 perfmon event select MSR.
F51H	3921	MSR_C8_PMON_CTR0	Package	Uncore C-box 8 perfmon counter MSR.
F52H	3922	MSR_C8_PMON_EVNT_SEL1	Package	Uncore C-box 8 perfmon event select MSR.
F53H	3923	MSR_C8_PMON_CTR1	Package	Uncore C-box 8 perfmon counter MSR.
F54H	3924	MSR_C8_PMON_EVNT_SEL2	Package	Uncore C-box 8 perfmon event select MSR.
F55H	3925	MSR_C8_PMON_CTR2	Package	Uncore C-box 8 perfmon counter MSR.
F56H	3926	MSR_C8_PMON_EVNT_SEL3	Package	Uncore C-box 8 perfmon event select MSR.
F57H	3927	MSR_C8_PMON_CTR3	Package	Uncore C-box 8 perfmon counter MSR.
F58H	3928	MSR_C8_PMON_EVNT_SEL4	Package	Uncore C-box 8 perfmon event select MSR.
F59H	3929	MSR_C8_PMON_CTR4	Package	Uncore C-box 8 perfmon counter MSR.
F5AH	3930	MSR_C8_PMON_EVNT_SEL5	Package	Uncore C-box 8 perfmon event select MSR.
F5BH	3931	MSR_C8_PMON_CTR5	Package	Uncore C-box 8 perfmon counter MSR.

Table 2-19. Additional MSRs Supported by Intel® Xeon® Processor E7 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
FC0H	4032	MSR_C9_PMON_BOX_CTRL	Package	Uncore C-box 9 perfmon local box control MSR.
FC1H	4033	MSR_C9_PMON_BOX_STATUS	Package	Uncore C-box 9 perfmon local box status MSR.
FC2H	4034	MSR_C9_PMON_BOX_OVF_CTRL	Package	Uncore C-box 9 perfmon local box overflow control MSR.
FD0H	4048	MSR_C9_PMON_EVNT_SEL0	Package	Uncore C-box 9 perfmon event select MSR.
FD1H	4049	MSR_C9_PMON_CTR0	Package	Uncore C-box 9 perfmon counter MSR.
FD2H	4050	MSR_C9_PMON_EVNT_SEL1	Package	Uncore C-box 9 perfmon event select MSR.
FD3H	4051	MSR_C9_PMON_CTR1	Package	Uncore C-box 9 perfmon counter MSR.
FD4H	4052	MSR_C9_PMON_EVNT_SEL2	Package	Uncore C-box 9 perfmon event select MSR.
FD5H	4053	MSR_C9_PMON_CTR2	Package	Uncore C-box 9 perfmon counter MSR.
FD6H	4054	MSR_C9_PMON_EVNT_SEL3	Package	Uncore C-box 9 perfmon event select MSR.
FD7H	4055	MSR_C9_PMON_CTR3	Package	Uncore C-box 9 perfmon counter MSR.
FD8H	4056	MSR_C9_PMON_EVNT_SEL4	Package	Uncore C-box 9 perfmon event select MSR.
FD9H	4057	MSR_C9_PMON_CTR4	Package	Uncore C-box 9 perfmon counter MSR.
FDAH	4058	MSR_C9_PMON_EVNT_SEL5	Package	Uncore C-box 9 perfmon event select MSR.
FDBH	4059	MSR_C9_PMON_CTR5	Package	Uncore C-box 9 perfmon counter MSR.

2.11 MSRS IN 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 with DisplayFamily_DisplayModel of 06_2AH, 06_2DH, see Table 2-1. Additional MSRs specific to 06_2AH are listed in Table 2-21.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_ SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination" and Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.17, "Time-Stamp Counter" and see Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
17H	23	IA32_PLATFORM_ID	Package	Platform ID (R) See Table 2-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location" and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	SMI Counter (R/O)
		31:0		SMI Count (R/O) Count SMIs.
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		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)
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	BIOS Update Signature ID (RO) See Table 2-2.
C1H	193	IA32_PMC0	Thread	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Thread	Performance Counter Register See Table 2-2.
C3H	195	IA32_PMC2	Thread	Performance Counter Register See Table 2-2.
C4H	196	IA32_PMC3	Thread	Performance Counter Register See Table 2-2.
C5H	197	IA32_PMC4	Core	Performance Counter Register (if core not shared by threads)
C6H	198	IA32_PMC5	Core	Performance Counter Register (if core not shared by threads)
C7H	199	IA32_PMC6	Core	Performance Counter Register (if core not shared by threads)
C8H	200	IA32_PMC7	Core	Performance Counter Register (if core not shared by threads)
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.
		7:0		Reserved

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	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.
		29	Package	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.
		39:30		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	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.
		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-sate 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

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		28		Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.
		63:29		Reserved
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	Power Management IO Redirection in C-state (R/W) See http://biosbits.org.
		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[bit10]: 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
E7H	231	IA32_MPERF	Thread	Maximum Performance Frequency Clock Count (RW) See Table 2-2.
E8H	232	IA32_APERF	Thread	Actual Performance Frequency Clock Count (RW) See Table 2-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 2-2.
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status
		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
186H	390	IA32_PERFEVTSEL0	Thread	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 2-2.
188H	392	IA32_PERFEVTSEL2	Thread	See Table 2-2.
189H	393	IA32_PERFEVTSEL3	Thread	See Table 2-2.
18AH	394	IA32_PERFEVTSEL4	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 4.
18BH	395	IA32_PERFEVTSEL5	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 5.
18CH	396	IA32_PERFEVTSEL6	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 6.
18DH	397	IA32_PERFEVTSEL7	Core	See Table 2-2. If CPUID.0AH:EAX[15:8] > 7.
198H	408	IA32_PERF_STATUS	Package	See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15:0		Current Performance State Value
		63:16		Reserved
198H	408	MSR_PERF_STATUS	Package	Performance Status
		47:32		Core Voltage (R/O) P-state core voltage can be computed by MSR_PERF_STATUS[37:32] * (float) 1/(2^13).
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		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
19BH	411	IA32_THERM_INTERRUPT	Core	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (RO) See Table 2-2.
		1		Thermal Status Log (R/WC0) See Table 2-2.
		2		PROTCHOT # or FORCEPR# Status (RO) See Table 2-2.
		3		PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.
		4		Critical Temperature Status (RO) See Table 2-2.
		5		Critical Temperature Status Log (R/WC0) See Table 2-2.
		6		Thermal Threshold #1 Status (RO) See Table 2-2.
		7		Thermal Threshold #1 Log (R/WC0) See Table 2-2.
		8		Thermal Threshold #2 Status (RO) See Table 2-2.
		9		Thermal Threshold #2 Log (R/WC0) See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		10		Power Limitation Status (RO) See Table 2-2.
		11		Power Limitation Log (R/WC0) See Table 2-2.
		15:12		Reserved
		22:16		Digital Readout (RO) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (RO) See Table 2-2.
		31		Reading Valid (RO) See Table 2-2.
		63:32		Reserved
1A0H	416	IA32_MISC_ENABLE		Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.
		0	Thread	Fast-Strings Enable See Table 2-2
		6:1		Reserved
		7	Thread	Performance Monitoring Available (R) See Table 2-2.
		10:8		Reserved
		11	Thread	Branch Trace Storage Unavailable (RO) See Table 2-2.
		12	Thread	Processor Event Based Sampling Unavailable (RO) See Table 2-2.
		15:13		Reserved
		16	Package	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.
		18	Thread	ENABLE MONITOR FSM (R/W) See Table 2-2.
		21:19		Reserved
		22	Thread	Limit CPUID Maxval (R/W) See Table 2-2.
		23	Thread	xTPR Message Disable (R/W) See Table 2-2.
		33:24		Reserved
		34	Thread	XD Bit Disable (R/W) See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		37:35		Reserved
		38	Package	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.
		63:39		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Unique	Temperature Target
		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
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	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.
		1	Core	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).
		2	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.
		3	Core	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.
		63:4		Reserved
1A6H	422	MSR_OFFCORE_RSP_0	Thread	Offcore Response Event Select Register (R/W)
1A7H	422	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)
1AAH	426	MSR_MISC_PWR_MGMT		Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 2-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 2-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 2-2.
1C8H	456	MSR_LBR_SELECT	Thread	Last Branch Record Filtering Select Register (R/W) See Section 17.9.2, "Filtering of Last Branch Records."
		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
1C9H	457	MSR_LASTBRANCH_TOS	Thread	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).
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W) See Table 2-2.
		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
		63:15		Reserved
1DDH	477	MSR_LER_FROM_LIP	Thread	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.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1DEH	478	MSR_LER_TO_LIP	Thread	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.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
1FCH	508	MSR_POWER_CTL	Core	See http://biosbits.org.
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 2-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 2-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 2-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 2-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 2-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 2-2.
277H	631	IA32_PAT	Thread	See Table 2-2.
280H	640	IA32_MC0_CTL2	Core	See Table 2-2.
281H	641	IA32_MC1_CTL2	Core	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	Always 0 (CMCI not supported).
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Thread	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
		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
38DH	909	IA32_FIXED_CTR_CTRL	Thread	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS		See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
		0	Thread	Ovf_PMC0
		1	Thread	Ovf_PMC1
		2	Thread	Ovf_PMC2
		3	Thread	Ovf_PMC3
		4	Core	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		5	Core	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)
		6	Core	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)
		7	Core	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)
		31:8		Reserved
		32	Thread	Ovf_FixedCtr0
		33	Thread	Ovf_FixedCtr1
		34	Thread	Ovf_FixedCtr2
		60:35		Reserved
		61	Thread	Ovf_Uncore
		62	Thread	Ovf_BufDSSAVE
		63	Thread	CondChgd
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
		0	Thread	Set 1 to enable PMC0 to count.
		1	Thread	Set 1 to enable PMC1 to count.
		2	Thread	Set 1 to enable PMC2 to count.
		3	Thread	Set 1 to enable PMC3 to count.
		4	Core	Set 1 to enable PMC4 to count (if CPUID.0AH:EAX[15:8] > 4).
		5	Core	Set 1 to enable PMC5 to count (if CPUID.0AH:EAX[15:8] > 5).
		6	Core	Set 1 to enable PMC6 to count (if CPUID.0AH:EAX[15:8] > 6).
		7	Core	Set 1 to enable PMC7 to count (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to enable FixedCtr0 to count.
		33	Thread	Set 1 to enable FixedCtr1 to count.
		34	Thread	Set 1 to enable FixedCtr2 to count.
		63:35		Reserved
390H	912	IA32_PERF_GLOBAL_OVF_CTRL		See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
		0	Thread	Set 1 to clear Ovf_PMC0.
		1	Thread	Set 1 to clear Ovf_PMC1.
		2	Thread	Set 1 to clear Ovf_PMC2.
		3	Thread	Set 1 to clear Ovf_PMC3.
		4	Core	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).
		5	Core	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).
		6	Core	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7	Core	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to clear Ovf_FixedCtr0.
		33	Thread	Set 1 to clear Ovf_FixedCtr1.
		34	Thread	Set 1 to clear Ovf_FixedCtr2.
		60:35		Reserved
		61	Thread	Set 1 to clear Ovf_Uncore.
		62	Thread	Set 1 to clear Ovf_BufDSSAVE.
		63	Thread	Set 1 to clear CondChgd.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Section 19.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		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)
3F6H	1014	MSR_PEBS_LD_LAT	Thread	See Section 19.3.1.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
3FEH	1022	MSR_CORE_C7_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
400H	1024	IA32_MC0_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
402H	1026	IA32_MC0_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
403H	1027	IA32_MC0_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
406H	1030	IA32_MC1_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
407H	1031	IA32_MC1_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40BH	1035	IA32_MC2_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40FH	1039	IA32_MC3_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
410H	1040	IA32_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
		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.
		63:2		Reserved
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
480H	1152	IA32_VMX_BASIC	Thread	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTLS	Thread	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTLS	Thread	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTLS	Thread	Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Thread	Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
485H	1157	IA32_VMX_MISC	Thread	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Thread	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Thread	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Thread	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Thread	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Thread	Capability Reporting Register of EPT and VPID (R/O) See Table 2-2
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	Thread	Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	Thread	Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	Thread	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	Thread	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2
4C1H	1217	IA32_A_PMC0	Thread	See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
4C2H	1218	IA32_A_PMC1	Thread	See Table 2-2.
4C3H	1219	IA32_A_PMC2	Thread	See Table 2-2.
4C4H	1220	IA32_A_PMC3	Thread	See Table 2-2.
4C5H	1221	IA32_A_PMC4	Core	See Table 2-2.
4C6H	1222	IA32_A_PMC5	Core	See Table 2-2.
4C7H	1223	IA32_A_PMC6	Core	See Table 2-2.
4C8H	1224	IA32_A_PMC7	Core	See Table 2-2.
600H	1536	IA32_DS_AREA	Thread	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 14.10.1, "RAPL Interfaces."
60AH	1546	MSR_PKGC3_IRTL	Package	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.
		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-sate management.
		63:16		Reserved

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
60BH	1547	MSR_PKGC6_IRTL	Package	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.
		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-sate management.
		63:16		Reserved
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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.
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 14.10.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 14.10.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameters (R/W) See Section 14.10.3, "Package RAPL Domain."
638H	1592	MSR_PP0_POWER_LIMIT	Package	PP0 RAPL Power Limit Control (R/W) See Section 14.10.4, "PP0/PP1 RAPL Domains."

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	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 17.9.1 and record format in Section 17.4.8.1.
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	Thread	Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	Thread	Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	Thread	Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	Thread	Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	Thread	Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	Thread	Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	Thread	Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	Thread	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.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	Thread	Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	Thread	Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	Thread	Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	Thread	Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	Thread	Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	Thread	Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	Thread	Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	Thread	Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	Thread	Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	Thread	Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	Thread	Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	Thread	Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	Thread	Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	Thread	Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	Thread	Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6E0H	1760	IA32_TSC_DEADLINE	Thread	See Table 2-2.
802H-83FH	2050-2111	X2APIC MSRs	Thread	See Table 2-2.
C000_0080H		IA32_EFER	Thread	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Thread	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	IA-32e Mode System Call Target Address (R/W) See Table 2-2.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C000_0084H		IA32_FMASK	Thread	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	Thread	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	Swap Target of BASE Address of GS (R/W) See Table 2-2.
C000_0103H		IA32_TSC_AUX	Thread	AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 17.17.2, "IA32_TSC_AUX Register and RDTSCP Support."

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 with DisplayFamily_DisplayModel of 06_2AH; see Table 2-1.

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
60CH	1548	MSR_PKGC7_IRTL	Package	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.
		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-sate management.
		63:16		Reserved
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
63AH	1594	MSR_PP0_POLICY	Package	PP0 Balance Policy (R/W) See Section 14.10.4, "PP0/PP1 RAPL Domains."
640H	1600	MSR_PP1_POWER_LIMIT	Package	PP1 RAPL Power Limit Control (R/W) See Section 14.10.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	PP1 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
642H	1602	MSR_PP1_POLICY	Package	PP1 Balance Policy (R/W) See Section 14.10.4, "PP0/PP1 RAPL Domains."
See Table 2-20, Table 2-21, and Table 2-22 for MSR definitions applicable to processors with CPUID signature 06_2AH.				

Table 2-22 lists the MSRs of uncore PMU for Intel processors with CPUID signature 06_2AH.

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		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
392H	914	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Report the number of C-Box units with performance counters, including processor cores and processor graphics.
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSEL0	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, Counter 1 Event Select MSR

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
700H	1792	MSR_UNC_CBO_0_PERFEVTSEL0	Package	Uncore C-Box 0, Counter 0 Event Select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
702H	1794	MSR_UNC_CBO_0_PERFEVTSEL2	Package	Uncore C-Box 0, Counter 2 Event Select MSR
703H	1795	MSR_UNC_CBO_0_PERFEVTSEL3	Package	Uncore C-Box 0, Counter 3 Event Select MSR
705H	1797	MSR_UNC_CBO_0_UNIT_STATUS	Package	Uncore C-Box 0, Unit Status for Counter 0-3
706H	1798	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, Performance Counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
708H	1800	MSR_UNC_CBO_0_PERFCTR2	Package	Uncore C-Box 0, Performance Counter 2
709H	1801	MSR_UNC_CBO_0_PERFCTR3	Package	Uncore C-Box 0, Performance Counter 3
710H	1808	MSR_UNC_CBO_1_PERFEVTSEL0	Package	Uncore C-Box 1, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
712H	1810	MSR_UNC_CBO_1_PERFEVTSEL2	Package	Uncore C-Box 1, Counter 2 Event Select MSR
713H	1811	MSR_UNC_CBO_1_PERFEVTSEL3	Package	Uncore C-Box 1, Counter 3 Event Select MSR
715H	1813	MSR_UNC_CBO_1_UNIT_STATUS	Package	Uncore C-Box 1, Unit Status for Counter 0-3
716H	1814	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, Performance Counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
718H	1816	MSR_UNC_CBO_1_PERFCTR2	Package	Uncore C-Box 1, Performance Counter 2
719H	1817	MSR_UNC_CBO_1_PERFCTR3	Package	Uncore C-Box 1, Performance Counter 3
720H	1824	MSR_UNC_CBO_2_PERFEVTSEL0	Package	Uncore C-Box 2, Counter 0 Event Select MSR
721H	1825	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
722H	1826	MSR_UNC_CBO_2_PERFEVTSEL2	Package	Uncore C-Box 2, Counter 2 Event Select MSR
723H	1827	MSR_UNC_CBO_2_PERFEVTSEL3	Package	Uncore C-Box 2, Counter 3 Event Select MSR
725H	1829	MSR_UNC_CBO_2_UNIT_STATUS	Package	Uncore C-Box 2, Unit Status for Counter 0-3
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, Performance Counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
728H	1832	MSR_UNC_CBO_3_PERFCTR2	Package	Uncore C-Box 3, Performance Counter 2
729H	1833	MSR_UNC_CBO_3_PERFCTR3	Package	Uncore C-Box 3, Performance Counter 3
730H	1840	MSR_UNC_CBO_3_PERFEVTSEL0	Package	Uncore C-Box 3, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
732H	1842	MSR_UNC_CBO_3_PERFEVTSEL2	Package	Uncore C-Box 3, Counter 2 Event Select MSR
733H	1843	MSR_UNC_CBO_3_PERFEVTSEL3	Package	Uncore C-Box 3, counter 3 Event Select MSR
735H	1845	MSR_UNC_CBO_3_UNIT_STATUS	Package	Uncore C-Box 3, Unit Status for Counter 0-3
736H	1846	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, Performance Counter 0
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
738H	1848	MSR_UNC_CBO_3_PERFCTR2	Package	Uncore C-Box 3, Performance Counter 2
739H	1849	MSR_UNC_CBO_3_PERFCTR3	Package	Uncore C-Box 3, Performance Counter 3
740H	1856	MSR_UNC_CBO_4_PERFEVTSEL0	Package	Uncore C-Box 4, Counter 0 Event Select MSR

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
741H	1857	MSR_UNC_CBO_4_PERFEVTSEL1	Package	Uncore C-Box 4, Counter 1 Event Select MSR
742H	1858	MSR_UNC_CBO_4_PERFEVTSEL2	Package	Uncore C-Box 4, Counter 2 Event Select MSR
743H	1859	MSR_UNC_CBO_4_PERFEVTSEL3	Package	Uncore C-Box 4, Counter 3 Event Select MSR
745H	1861	MSR_UNC_CBO_4_UNIT_STATUS	Package	Uncore C-Box 4, Unit status for Counter 0-3
746H	1862	MSR_UNC_CBO_4_PERFCTR0	Package	Uncore C-Box 4, Performance Counter 0
747H	1863	MSR_UNC_CBO_4_PERFCTR1	Package	Uncore C-Box 4, Performance Counter 1
748H	1864	MSR_UNC_CBO_4_PERFCTR2	Package	Uncore C-Box 4, Performance Counter 2
749H	1865	MSR_UNC_CBO_4_PERFCTR3	Package	Uncore C-Box 4, Performance Counter 3

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 the Sandy Bridge microarchitecture. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2DH, and also support MSRs listed in Table 2-20 and Table 2-24.

Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
17FH	383	MSR_ERROR_CONTROL	Package	MC Bank Error Configuration (R/W)
		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
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 cores active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 cores active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 cores active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 cores active.

Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 cores active.
		55:48	Package	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 cores active.
		63:56	Package	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 cores active.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
39CH	924	MSR_PEBS_NUM_ALT	Package	ENABLE_PEBS_NUM_ALT (RW)
		0		ENABLE_PEBS_NUM_ALT (RW) 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.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
416H	1046	IA32_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	IA32_MC5_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	IA32_MC6_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
41AH	1050	IA32_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	IA32_MC6_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
41DH	1053	IA32_MC7_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
41EH	1054	IA32_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	IA32_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	IA32_MC8_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
422H	1058	IA32_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	IA32_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
425H	1061	IA32_MC9_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
426H	1062	IA32_MC9_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
427H	1063	IA32_MC9_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
429H	1065	IA32_MC10_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
42AH	1066	IA32_MC10_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42BH	1067	IA32_MC10_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	IA32_MC11_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
42EH	1070	IA32_MC11_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42FH	1071	IA32_MC11_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
431H	1073	IA32_MC12_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
432H	1074	IA32_MC12_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
433H	1075	IA32_MC12_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
435H	1077	IA32_MC13_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
436H	1078	IA32_MC13_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
437H	1079	IA32_MC13_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
439H	1081	IA32_MC14_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
43AH	1082	IA32_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	IA32_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."

Table 2-23. Selected MSRs Supported by Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
43DH	1085	IA32_MC15_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
43EH	1086	IA32_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43FH	1087	IA32_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
441H	1089	IA32_MC16_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
442H	1090	IA32_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
443H	1091	IA32_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	IA32_MC17_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
446H	1094	IA32_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	IA32_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	IA32_MC18_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
44AH	1098	IA32_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	IA32_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	IA32_MC19_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS" and Chapter 16.
44EH	1102	IA32_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	IA32_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
613H	1555	MSR_PKG_PERF_STATUS	Package	Package RAPL Perf Status (R/O)
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
See Table 2-20, Table 2-23, and Table 2-24 for MSR definitions applicable to processors with CPUID signature 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 detail of the uncore PMU, refer to Intel Xeon Processor E5 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2DH

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C08H	3080	MSR_U_PMON_UCLK_FIXED_CTL	Package	Uncore U-box UCLK Fixed Counter Control
C09H	3081	MSR_U_PMON_UCLK_FIXED_CTR	Package	Uncore U-box UCLK Fixed Counter
C10H	3088	MSR_U_PMON_EVNTSEL0	Package	Uncore U-box Perfmon Event Select for U-box Counter 0
C11H	3089	MSR_U_PMON_EVNTSEL1	Package	Uncore U-box Perfmon Event Select for U-box Counter 1
C16H	3094	MSR_U_PMON_CTR0	Package	Uncore U-box Perfmon Counter 0
C17H	3095	MSR_U_PMON_CTR1	Package	Uncore U-box Perfmon Counter 1
C24H	3108	MSR_PCU_PMON_BOX_CTL	Package	Uncore PCU Perfmon for PCU-box-wide Control
C30H	3120	MSR_PCU_PMON_EVNTSEL0	Package	Uncore PCU Perfmon Event Select for PCU Counter 0
C31H	3121	MSR_PCU_PMON_EVNTSEL1	Package	Uncore PCU Perfmon Event Select for PCU Counter 1
C32H	3122	MSR_PCU_PMON_EVNTSEL2	Package	Uncore PCU Perfmon Event Select for PCU Counter 2
C33H	3123	MSR_PCU_PMON_EVNTSEL3	Package	Uncore PCU Perfmon Event Select for PCU Counter 3
C34H	3124	MSR_PCU_PMON_BOX_FILTER	Package	Uncore PCU Perfmon box-wide Filter
C36H	3126	MSR_PCU_PMON_CTR0	Package	Uncore PCU Perfmon Counter 0
C37H	3127	MSR_PCU_PMON_CTR1	Package	Uncore PCU Perfmon Counter 1
C38H	3128	MSR_PCU_PMON_CTR2	Package	Uncore PCU Perfmon Counter 2
C39H	3129	MSR_PCU_PMON_CTR3	Package	Uncore PCU Perfmon Counter 3
D04H	3332	MSR_C0_PMON_BOX_CTL	Package	Uncore C-box 0 Perfmon Local Box Wide Control
D10H	3344	MSR_C0_PMON_EVNTSEL0	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 0
D11H	3345	MSR_C0_PMON_EVNTSEL1	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 1
D12H	3346	MSR_C0_PMON_EVNTSEL2	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 2
D13H	3347	MSR_C0_PMON_EVNTSEL3	Package	Uncore C-box 0 Perfmon Event Select for C-box 0 Counter 3
D14H	3348	MSR_C0_PMON_BOX_FILTER	Package	Uncore C-box 0 Perfmon Box Wide Filter
D16H	3350	MSR_C0_PMON_CTR0	Package	Uncore C-box 0 Perfmon Counter 0
D17H	3351	MSR_C0_PMON_CTR1	Package	Uncore C-box 0 Perfmon Counter 1
D18H	3352	MSR_C0_PMON_CTR2	Package	Uncore C-box 0 Perfmon Counter 2
D19H	3353	MSR_C0_PMON_CTR3	Package	Uncore C-box 0 Perfmon Counter 3
D24H	3364	MSR_C1_PMON_BOX_CTL	Package	Uncore C-box 1 Perfmon Local Box Wide Control

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D30H	3376	MSR_C1_PMON_EVNTSEL0	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 0
D31H	3377	MSR_C1_PMON_EVNTSEL1	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 1
D32H	3378	MSR_C1_PMON_EVNTSEL2	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 2
D33H	3379	MSR_C1_PMON_EVNTSEL3	Package	Uncore C-box 1 Perfmon Event Select for C-box 1 Counter 3
D34H	3380	MSR_C1_PMON_BOX_FILTER	Package	Uncore C-box 1 Perfmon Box Wide Filter
D36H	3382	MSR_C1_PMON_CTR0	Package	Uncore C-box 1 Perfmon Counter 0
D37H	3383	MSR_C1_PMON_CTR1	Package	Uncore C-box 1 Perfmon Counter 1
D38H	3384	MSR_C1_PMON_CTR2	Package	Uncore C-box 1 Perfmon Counter 2
D39H	3385	MSR_C1_PMON_CTR3	Package	Uncore C-box 1 Perfmon Counter 3
D44H	3396	MSR_C2_PMON_BOX_CTL	Package	Uncore C-box 2 Perfmon Local Box Wide Control
D50H	3408	MSR_C2_PMON_EVNTSEL0	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 0
D51H	3409	MSR_C2_PMON_EVNTSEL1	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 1
D52H	3410	MSR_C2_PMON_EVNTSEL2	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 2
D53H	3411	MSR_C2_PMON_EVNTSEL3	Package	Uncore C-box 2 Perfmon Event Select for C-box 2 Counter 3
D54H	3412	MSR_C2_PMON_BOX_FILTER	Package	Uncore C-box 2 Perfmon Box Wide Filter
D56H	3414	MSR_C2_PMON_CTR0	Package	Uncore C-box 2 Perfmon Counter 0
D57H	3415	MSR_C2_PMON_CTR1	Package	Uncore C-box 2 Perfmon Counter 1
D58H	3416	MSR_C2_PMON_CTR2	Package	Uncore C-box 2 Perfmon Counter 2
D59H	3417	MSR_C2_PMON_CTR3	Package	Uncore C-box 2 Perfmon Counter 3
D64H	3428	MSR_C3_PMON_BOX_CTL	Package	Uncore C-box 3 Perfmon Local Box Wide Control
D70H	3440	MSR_C3_PMON_EVNTSEL0	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 0
D71H	3441	MSR_C3_PMON_EVNTSEL1	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 1
D72H	3442	MSR_C3_PMON_EVNTSEL2	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 2
D73H	3443	MSR_C3_PMON_EVNTSEL3	Package	Uncore C-box 3 Perfmon Event Select for C-box 3 Counter 3
D74H	3444	MSR_C3_PMON_BOX_FILTER	Package	Uncore C-box 3 Perfmon Box Wide Filter
D76H	3446	MSR_C3_PMON_CTR0	Package	Uncore C-box 3 Perfmon Counter 0
D77H	3447	MSR_C3_PMON_CTR1	Package	Uncore C-box 3 Perfmon Counter 1
D78H	3448	MSR_C3_PMON_CTR2	Package	Uncore C-box 3 Perfmon Counter 2
D79H	3449	MSR_C3_PMON_CTR3	Package	Uncore C-box 3 Perfmon Counter 3

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D84H	3460	MSR_C4_PMON_BOX_CTL	Package	Uncore C-box 4 Perfmon Local Box Wide Control
D90H	3472	MSR_C4_PMON_EVNTSEL0	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 0
D91H	3473	MSR_C4_PMON_EVNTSEL1	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 1
D92H	3474	MSR_C4_PMON_EVNTSEL2	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 2
D93H	3475	MSR_C4_PMON_EVNTSEL3	Package	Uncore C-box 4 Perfmon Event Select for C-box 4 Counter 3
D94H	3476	MSR_C4_PMON_BOX_FILTER	Package	Uncore C-box 4 Perfmon Box Wide Filter
D96H	3478	MSR_C4_PMON_CTR0	Package	Uncore C-box 4 Perfmon Counter 0
D97H	3479	MSR_C4_PMON_CTR1	Package	Uncore C-box 4 Perfmon Counter 1
D98H	3480	MSR_C4_PMON_CTR2	Package	Uncore C-box 4 Perfmon Counter 2
D99H	3481	MSR_C4_PMON_CTR3	Package	Uncore C-box 4 Perfmon Counter 3
DA4H	3492	MSR_C5_PMON_BOX_CTL	Package	Uncore C-box 5 Perfmon Local Box Wide Control
DB0H	3504	MSR_C5_PMON_EVNTSEL0	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 0
DB1H	3505	MSR_C5_PMON_EVNTSEL1	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 1
DB2H	3506	MSR_C5_PMON_EVNTSEL2	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 2
DB3H	3507	MSR_C5_PMON_EVNTSEL3	Package	Uncore C-box 5 Perfmon Event Select for C-box 5 Counter 3
DB4H	3508	MSR_C5_PMON_BOX_FILTER	Package	Uncore C-box 5 Perfmon Box Wide Filter
DB6H	3510	MSR_C5_PMON_CTR0	Package	Uncore C-box 5 Perfmon Counter 0
DB7H	3511	MSR_C5_PMON_CTR1	Package	Uncore C-box 5 Perfmon Counter 1
DB8H	3512	MSR_C5_PMON_CTR2	Package	Uncore C-box 5 Perfmon Counter 2
DB9H	3513	MSR_C5_PMON_CTR3	Package	Uncore C-box 5 Perfmon Counter 3
DC4H	3524	MSR_C6_PMON_BOX_CTL	Package	Uncore C-box 6 Perfmon Local Box Wide Control
DD0H	3536	MSR_C6_PMON_EVNTSEL0	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 0
DD1H	3537	MSR_C6_PMON_EVNTSEL1	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 1
DD2H	3538	MSR_C6_PMON_EVNTSEL2	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 2
DD3H	3539	MSR_C6_PMON_EVNTSEL3	Package	Uncore C-box 6 Perfmon Event Select for C-box 6 Counter 3
DD4H	3540	MSR_C6_PMON_BOX_FILTER	Package	Uncore C-box 6 Perfmon Box Wide Filter
DD6H	3542	MSR_C6_PMON_CTR0	Package	Uncore C-box 6 Perfmon Counter 0
DD7H	3543	MSR_C6_PMON_CTR1	Package	Uncore C-box 6 Perfmon Counter 1
DD8H	3544	MSR_C6_PMON_CTR2	Package	Uncore C-box 6 Perfmon Counter 2

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DD9H	3545	MSR_C6_PMON_CTR3	Package	Uncore C-box 6 Perfmon Counter 3
DE4H	3556	MSR_C7_PMON_BOX_CTL	Package	Uncore C-box 7 Perfmon Local Box Wide Control
DF0H	3568	MSR_C7_PMON_EVNTSEL0	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 0
DF1H	3569	MSR_C7_PMON_EVNTSEL1	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 1
DF2H	3570	MSR_C7_PMON_EVNTSEL2	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 2
DF3H	3571	MSR_C7_PMON_EVNTSEL3	Package	Uncore C-box 7 Perfmon Event Select for C-box 7 Counter 3
DF4H	3572	MSR_C7_PMON_BOX_FILTER	Package	Uncore C-box 7 Perfmon Box Wide Filter
DF6H	3574	MSR_C7_PMON_CTR0	Package	Uncore C-box 7 Perfmon Counter 0
DF7H	3575	MSR_C7_PMON_CTR1	Package	Uncore C-box 7 Perfmon Counter 1
DF8H	3576	MSR_C7_PMON_CTR2	Package	Uncore C-box 7 Perfmon Counter 2
DF9H	3577	MSR_C7_PMON_CTR3	Package	Uncore C-box 7 Perfmon Counter 3

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 with DisplayFamily_DisplayModel of 06_3AH.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (Based on Ivy Bridge Microarchitecture)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	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.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (Based on Ivy Bridge Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		29	Package	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.
		31:30		Reserved
		32	Package	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.
		34:33	Package	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
		39:35		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		55:48	Package	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.
		63:56		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	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.
		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-sate 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.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (Based on Ivy Bridge Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		28		Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.
		63:29		Reserved
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	Base TDP Ratio (R/O)
		7:0		Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).
		63:8		Reserved
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O)
		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.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors (Based on Ivy Bridge Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		47		Reserved
		62:48		PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.
		63		Reserved
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O)
		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
		62:48		PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.
		63		Reserved
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	ConfigTDP Control (R/W)
		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
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W)
		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 CPUID signature 06_3AH.				

2.12.1 MSRs In 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 with DisplayFamily_DisplayModel 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 Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		22:16		Reserved
		23	Package	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.
		27:24		Reserved
		28	Package	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.

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		29	Package	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.
		30	Package	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.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	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.
		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-sate 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.

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		63:16		Reserved
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		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
17FH	383	MSR_ERROR_CONTROL	Package	MC Bank Error Configuration (R/W)
		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
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (RO) 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
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.
		15:8	Package	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.
		23:16	Package	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		31:24	Package	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.
		63:32		Reserved
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
296H	662	IA32_MC22_CTL2	Package	See Table 2-2.
297H	663	IA32_MC23_CTL2	Package	See Table 2-2.
298H	664	IA32_MC24_CTL2	Package	See Table 2-2.
299H	665	IA32_MC25_CTL2	Package	See Table 2-2.
29AH	666	IA32_MC26_CTL2	Package	See Table 2-2.
29BH	667	IA32_MC27_CTL2	Package	See Table 2-2.
29CH	668	IA32_MC28_CTL2	Package	See Table 2-2.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC5 reports MC errors from the Intel QPI module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC7 and MC 8 report MC errors from the two home agents.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC7 and MC 8 report MC errors from the two home agents.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	IA32_MC11_STATUS	Package	Bank MC11 reports MC errors from a specific channel of the integrated memory controller.
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
451H	1105	IA32_MC20_STATUS	Package	Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
458H	1112	IA32_MC22_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
459H	1113	IA32_MC22_STATUS	Package	
45AH	1114	IA32_MC22_ADDR	Package	
45BH	1115	IA32_MC22_MISC	Package	

Table 2-26. MSRs Supported by Intel® Xeon® Processors E5 v2 Product Family (based on Ivy Bridge-E microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
45CH	1116	IA32_MC23_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
45DH	1117	IA32_MC23_STATUS	Package	
45EH	1118	IA32_MC23_ADDR	Package	
45FH	1119	IA32_MC23_MISC	Package	
460H	1120	IA32_MC24_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
461H	1121	IA32_MC24_STATUS	Package	
462H	1122	IA32_MC24_ADDR	Package	
463H	1123	IA32_MC24_MISC	Package	
464H	1124	IA32_MC25_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
465H	1125	IA32_MC25_STATUS	Package	
466H	1126	IA32_MC25_ADDR	Package	
467H	1127	IA32_MC2MISC	Package	
468H	1128	IA32_MC26_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
469H	1129	IA32_MC26_STATUS	Package	
46AH	1130	IA32_MC26_ADDR	Package	
46BH	1131	IA32_MC26_MISC	Package	
46CH	1132	IA32_MC27_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
46DH	1133	IA32_MC27_STATUS	Package	
46EH	1134	IA32_MC27_ADDR	Package	
46FH	1135	IA32_MC27_MISC	Package	
470H	1136	IA32_MC28_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
471H	1137	IA32_MC28_STATUS	Package	
472H	1138	IA32_MC28_ADDR	Package	
473H	1139	IA32_MC28_MISC	Package	
613H	1555	MSR_PKG_PERF_STATUS	Package	Package RAPL Perf Status (R/O)
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
See Table 2-20, for other MSR definitions applicable to Intel Xeon processor E5 v2 with CPUID signature 06_3EH.				

2.12.2 Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family

Intel® Xeon® processor E7 v2 family (based on Ivy Bridge-E microarchitecture) with CPUID DisplayFamily_DisplayModel signature 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 Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		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
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		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
17AH	378	IA32_MCG_STATUS	Thread	Global Machine Check Status (R/WO)
		0		RIPV
		1		EIPV
		2		MCIP
		3		LMCE Signaled
		63:4		Reserved
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.
		15:8	Package	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10core active.

Table 2-27. Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		23:16	Package	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.
		31:24	Package	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.
		39:32	Package	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.
		47:40	Package	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.
		55:48	Package	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.
		62:56		Reserved
		63	Package	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).
29DH	669	IA32_MC29_CTL2	Package	See Table 2-2.
29EH	670	IA32_MC30_CTL2	Package	See Table 2-2.
29FH	671	IA32_MC31_CTL2	Package	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Section 19.3.1.1.1, "Processor Event Based Sampling (PEBS)."
		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
41BH	1051	IA32_MC6_MISC	Package	Misc MAC Information of Integrated I/O (R/O) See Section 15.3.2.4.
		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
474H	1140	IA32_MC29_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
475H	1141	IA32_MC29_STATUS	Package	
476H	1142	IA32_MC29_ADDR	Package	
477H	1143	IA32_MC29_MISC	Package	

Table 2-27. Additional MSRs Supported by Intel® Xeon® Processor E7 v2 Family with DisplayFamily_DisplayModel Signature 06_3EH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
478H	1144	IA32_MC30_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
479H	1145	IA32_MC30_STATUS	Package	
47AH	1146	IA32_MC30_ADDR	Package	
47BH	1147	IA32_MC30_MISC	Package	
47CH	1148	IA32_MC31_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.
47DH	1149	IA32_MC31_STATUS	Package	
47EH	1150	IA32_MC31_ADDR	Package	
47FH	1147	IA32_MC31_MISC	Package	
See Table 2-20, Table 2-26 for other MSR definitions applicable to Intel Xeon processor E7 v2 with CPUID signature 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 with DisplayFamily_DisplayModel of 06_3EH.

Table 2-28. Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C00H	3072	MSR_PMON_GLOBAL_CTL	Package	Uncore Perfmon Per-Socket Global Control
C01H	3073	MSR_PMON_GLOBAL_STATUS	Package	Uncore Perfmon Per-Socket Global Status
C06H	3078	MSR_PMON_GLOBAL_CONFIG	Package	Uncore Perfmon Per-Socket Global Configuration
C15H	3093	MSR_U_PMON_BOX_STATUS	Package	Uncore U-box Perfmon U-Box Wide Status
C35H	3125	MSR_PCU_PMON_BOX_STATUS	Package	Uncore PCU Perfmon Box Wide Status
D1AH	3354	MSR_C0_PMON_BOX_FILTER1	Package	Uncore C-Box 0 Perfmon Box Wide Filter1
D3AH	3386	MSR_C1_PMON_BOX_FILTER1	Package	Uncore C-Box 1 Perfmon Box Wide Filter1
D5AH	3418	MSR_C2_PMON_BOX_FILTER1	Package	Uncore C-Box 2 Perfmon Box Wide Filter1
D7AH	3450	MSR_C3_PMON_BOX_FILTER1	Package	Uncore C-Box 3 Perfmon Box Wide Filter1
D9AH	3482	MSR_C4_PMON_BOX_FILTER1	Package	Uncore C-Box 4 Perfmon Box Wide Filter1
DBAH	3514	MSR_C5_PMON_BOX_FILTER1	Package	Uncore C-Box 5 Perfmon Box Wide Filter1
DDAH	3546	MSR_C6_PMON_BOX_FILTER1	Package	Uncore C-Box 6 Perfmon Box Wide Filter1
DFAH	3578	MSR_C7_PMON_BOX_FILTER1	Package	Uncore C-Box 7 Perfmon Box Wide Filter1
E04H	3588	MSR_C8_PMON_BOX_CTL	Package	Uncore C-Box 8 Perfmon Local Box Wide Control

Table 2-28. Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E10H	3600	MSR_C8_PMON_EVNTSEL0	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 0
E11H	3601	MSR_C8_PMON_EVNTSEL1	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 1
E12H	3602	MSR_C8_PMON_EVNTSEL2	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 2
E13H	3603	MSR_C8_PMON_EVNTSEL3	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 3
E14H	3604	MSR_C8_PMON_BOX_FILTER	Package	Uncore C-Box 8 Perfmon Box Wide Filter
E16H	3606	MSR_C8_PMON_CTR0	Package	Uncore C-Box 8 Perfmon Counter 0
E17H	3607	MSR_C8_PMON_CTR1	Package	Uncore C-Box 8 Perfmon Counter 1
E18H	3608	MSR_C8_PMON_CTR2	Package	Uncore C-Box 8 Perfmon Counter 2
E19H	3609	MSR_C8_PMON_CTR3	Package	Uncore C-Box 8 Perfmon Counter 3
E1AH	3610	MSR_C8_PMON_BOX_FILTER1	Package	Uncore C-Box 8 Perfmon Box Wide Filter1
E24H	3620	MSR_C9_PMON_BOX_CTL	Package	Uncore C-Box 9 Perfmon Local Box Wide Control
E30H	3632	MSR_C9_PMON_EVNTSEL0	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 0
E31H	3633	MSR_C9_PMON_EVNTSEL1	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 1
E32H	3634	MSR_C9_PMON_EVNTSEL2	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 2
E33H	3635	MSR_C9_PMON_EVNTSEL3	Package	Uncore C-Box 9 Perfmon Event Select for C-box 9 Counter 3
E34H	3636	MSR_C9_PMON_BOX_FILTER	Package	Uncore C-Box 9 Perfmon Box Wide Filter
E36H	3638	MSR_C9_PMON_CTR0	Package	Uncore C-Box 9 Perfmon Counter 0
E37H	3639	MSR_C9_PMON_CTR1	Package	Uncore C-Box 9 Perfmon Counter 1
E38H	3640	MSR_C9_PMON_CTR2	Package	Uncore C-Box 9 Perfmon Counter 2
E39H	3641	MSR_C9_PMON_CTR3	Package	Uncore C-Box 9 Perfmon Counter 3
E3AH	3642	MSR_C9_PMON_BOX_FILTER1	Package	Uncore C-Box 9 Perfmon Box Wide Filter1
E44H	3652	MSR_C10_PMON_BOX_CTL	Package	Uncore C-Box 10 Perfmon Local Box Wide Control
E50H	3664	MSR_C10_PMON_EVNTSEL0	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 0
E51H	3665	MSR_C10_PMON_EVNTSEL1	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 1
E52H	3666	MSR_C10_PMON_EVNTSEL2	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 2
E53H	3667	MSR_C10_PMON_EVNTSEL3	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 3
E54H	3668	MSR_C10_PMON_BOX_FILTER	Package	Uncore C-Box 10 Perfmon Box Wide Filter
E56H	3670	MSR_C10_PMON_CTR0	Package	Uncore C-Box 10 Perfmon Counter 0
E57H	3671	MSR_C10_PMON_CTR1	Package	Uncore C-Box 10 Perfmon Counter 1

Table 2-28. Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E58H	3672	MSR_C10_PMON_CTR2	Package	Uncore C-Box 10 Perfmon Counter 2
E59H	3673	MSR_C10_PMON_CTR3	Package	Uncore C-Box 10 Perfmon Counter 3
E5AH	3674	MSR_C10_PMON_BOX_FILTER1	Package	Uncore C-Box 10 Perfmon Box Wide Filter1
E64H	3684	MSR_C11_PMON_BOX_CTL	Package	Uncore C-Box 11 Perfmon Local Box Wide Control
E70H	3696	MSR_C11_PMON_EVNTSEL0	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 0
E71H	3697	MSR_C11_PMON_EVNTSEL1	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 1
E72H	3698	MSR_C11_PMON_EVNTSEL2	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 2
E73H	3699	MSR_C11_PMON_EVNTSEL3	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 3
E74H	3700	MSR_C11_PMON_BOX_FILTER	Package	Uncore C-Box 11 Perfmon Box Wide Filter
E76H	3702	MSR_C11_PMON_CTR0	Package	Uncore C-Box 11 Perfmon Counter 0
E77H	3703	MSR_C11_PMON_CTR1	Package	Uncore C-Box 11 Perfmon Counter 1
E78H	3704	MSR_C11_PMON_CTR2	Package	Uncore C-Box 11 Perfmon Counter 2
E79H	3705	MSR_C11_PMON_CTR3	Package	Uncore C-Box 11 Perfmon Counter 3
E7AH	3706	MSR_C11_PMON_BOX_FILTER1	Package	Uncore C-Box 11 Perfmon Box Wide Filter1
E84H	3716	MSR_C12_PMON_BOX_CTL	Package	Uncore C-Box 12 Perfmon Local Box Wide Control
E90H	3728	MSR_C12_PMON_EVNTSEL0	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 0
E91H	3729	MSR_C12_PMON_EVNTSEL1	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 1
E92H	3730	MSR_C12_PMON_EVNTSEL2	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 2
E93H	3731	MSR_C12_PMON_EVNTSEL3	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 3
E94H	3732	MSR_C12_PMON_BOX_FILTER	Package	Uncore C-Box 12 Perfmon Box Wide Filter
E96H	3734	MSR_C12_PMON_CTR0	Package	Uncore C-Box 12 Perfmon Counter 0
E97H	3735	MSR_C12_PMON_CTR1	Package	Uncore C-Box 12 Perfmon Counter 1
E98H	3736	MSR_C12_PMON_CTR2	Package	Uncore C-Box 12 Perfmon Counter 2
E99H	3737	MSR_C12_PMON_CTR3	Package	Uncore C-Box 12 Perfmon Counter 3
E9AH	3738	MSR_C12_PMON_BOX_FILTER1	Package	Uncore C-Box 12 Perfmon Box Wide Filter1
EA4H	3748	MSR_C13_PMON_BOX_CTL	Package	Uncore C-Box 13 Perfmon Local Box Wide Control
EB0H	3760	MSR_C13_PMON_EVNTSEL0	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 0
EB1H	3761	MSR_C13_PMON_EVNTSEL1	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 1
EB2H	3762	MSR_C13_PMON_EVNTSEL2	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 2

Table 2-28. Uncore PMU MSRs in Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
EB3H	3763	MSR_C13_PMON_EVNTSEL3	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 3
EB4H	3764	MSR_C13_PMON_BOX_FILTER	Package	Uncore C-Box 13 Perfmon Box Wide Filter
EB6H	3766	MSR_C13_PMON_CTR0	Package	Uncore C-Box 13 Perfmon Counter 0
EB7H	3767	MSR_C13_PMON_CTR1	Package	Uncore C-Box 13 Perfmon Counter 1
EB8H	3768	MSR_C13_PMON_CTR2	Package	Uncore C-Box 13 Perfmon Counter 2
EB9H	3769	MSR_C13_PMON_CTR3	Package	Uncore C-Box 13 Perfmon Counter 3
EBAH	3770	MSR_C13_PMON_BOX_FILTER1	Package	Uncore C-Box 13 Perfmon Box Wide Filter1
EC4H	3780	MSR_C14_PMON_BOX_CTL	Package	Uncore C-Box 14 Perfmon Local Box Wide Control
ED0H	3792	MSR_C14_PMON_EVNTSEL0	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 0
ED1H	3793	MSR_C14_PMON_EVNTSEL1	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 1
ED2H	3794	MSR_C14_PMON_EVNTSEL2	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 2
ED3H	3795	MSR_C14_PMON_EVNTSEL3	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 3
ED4H	3796	MSR_C14_PMON_BOX_FILTER	Package	Uncore C-Box 14 Perfmon Box Wide Filter
ED6H	3798	MSR_C14_PMON_CTR0	Package	Uncore C-Box 14 Perfmon Counter 0
ED7H	3799	MSR_C14_PMON_CTR1	Package	Uncore C-Box 14 Perfmon Counter 1
ED8H	3800	MSR_C14_PMON_CTR2	Package	Uncore C-Box 14 Perfmon Counter 2
ED9H	3801	MSR_C14_PMON_CTR3	Package	Uncore C-Box 14 Perfmon Counter 3
EDAH	3802	MSR_C14_PMON_BOX_FILTER1	Package	Uncore C-Box 14 Perfmon Box Wide Filter1

2.13 MSRS IN THE 4TH GENERATION INTEL® CORE™ PROCESSORS (BASED ON HASWELL MICROARCHITECTURE)

The 4th generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200v3 product family (based on Haswell microarchitecture), with CPUID DisplayFamily_DisplayModel signature 06_3CH/06_45H/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 supersede 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 or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3BH	59	IA32_TSC_ADJUST	Thread	Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.

Table 2-29. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	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.
		29	Package	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.
		31:30		Reserved
		32	Package	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.
		34:33	Package	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
		39:35		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		55:48	Package	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.
		63:56		Reserved
186H	390	IA32_PERFEVTSEL0	Thread	Performance Event Select for Counter 0 (R/W) Supports all fields described inTable 2-2 and the fields below.
		32		IN_TX: See Section 19.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.

Table 2-29. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
187H	391	IA32_PERFEVTSEL1	Thread	Performance Event Select for Counter 1 (R/W) Supports all fields described inTable 2-2 and the fields below.
		32		IN_TX: See Section 19.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.
188H	392	IA32_PERFEVTSEL2	Thread	Performance Event Select for Counter 2 (R/W) Supports all fields described inTable 2-2 and the fields below.
		32		IN_TX: See Section 19.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.
		33		IN_TXCP: See Section 19.3.6.5.1. 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.
189H	393	IA32_PERFEVTSEL3	Thread	Performance Event Select for Counter 3 (R/W) Supports all fields described inTable 2-2 and the fields below.
		32		IN_TX: See Section 19.3.6.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.
1C8H	456	MSR_LBR_SELECT	Thread	Last Branch Record Filtering Select Register (R/W)
		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
		63:9		Reserved
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W) See Table 2-2.

Table 2-29. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
491H	1169	IA32_VMX_VMFUNC	Thread	Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.
60BH	1548	MSR_PKGC_IRTL1	Package	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.
		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-sate management.
		63:16		Reserved

Table 2-29. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
60CH	1548	MSR_PKGC_IRTL2	Package	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.
		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-sate management.
		63:16		Reserved
613H	1555	MSR_PKG_PERF_STATUS	Package	PKG Perf Status (R/O) See Section 14.10.3, "Package RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	Base TDP Ratio (R/O)
		7:0		Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).
		63:8		Reserved
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 Ratio and Power Level (R/O)
		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

Table 2-29. Additional MSRs Supported by Processors based on the Haswell or Haswell-E microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 Ratio and Power Level (R/O)
		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
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	ConfigTDP Control (R/W)
		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
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W)
		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
C80H	3200	IA32_DEBUG_INTERFACE	Package	Silicon Debug Feature Control (R/W) See Table 2-2.

2.13.1 MSRs in 4th Generation Intel® Core™ Processor Family (based on Haswell Microarchitecture)

Table 2-30 lists model-specific registers (MSRs) that are specific to 4th generation Intel® Core™ processor family and Intel® Xeon® processor E3-1200 v3 product family (based on Haswell microarchitecture). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3CH/06_45H/06_46H, see Table 2-1.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	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.
		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 signature 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
17DH	381	MSR_SMM_MCA_CAP	THREAD	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		Reserved

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		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
392H	914	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Encoded number of C-Box, derive value by "-1".
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSEL0	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb Unit, Counter 1 Event Select MSR
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		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
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb Unit, Counter 1 Event Select MSR
4E0H	1248	MSR_SMM_FEATURE_CONTROL	Package	Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.
		0		Lock (SMM-RWO) When set to '1' locks this register from further changes.
		1		Reserved

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
4E2H	1250	MSR_SMM_DELAYED	Package	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.
		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.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
4E3H	1251	MSR_SMM_BLOCKED	Package	SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.
		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.(EAX=0BH, ECX=PKG_LVL):EBX[15:0] can be updated.
		63:N		Reserved
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 14.10.1, "RAPL Interfaces."
		7:4	Package	Reserved

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 14.10.1, "RAPL Interfaces."
		63:20		Reserved
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
640H	1600	MSR_PP1_POWER_LIMIT	Package	PP1 RAPL Power Limit Control (R/W) See Section 14.10.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERGY_STATUS	Package	PP1 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
642H	1602	MSR_PP1_POLICY	Package	PP1 Balance Policy (R/W) See Section 14.10.4, "PP0/PP1 RAPL Domains."
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		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

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		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.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
6B0H	1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)
		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.
		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.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		11		Package-Level PL2 Power Limiting Status (RO) 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.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		63:30		Reserved
6B1H	1713	MSR_RING_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)
		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).

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		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.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		63:30		Reserved
700H	1792	MSR_UNC_CBO_0_PERFEVTSEL0	Package	Uncore C-Box 0, Counter 0 Event Select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
706H	1798	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, Performance Counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
710H	1808	MSR_UNC_CBO_1_PERFEVTSEL0	Package	Uncore C-Box 1, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
716H	1814	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, Performance Counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
720H	1824	MSR_UNC_CBO_2_PERFEVTSEL0	Package	Uncore C-Box 2, Counter 0 Event Select MSR
721H	1824	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, Performance Counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
730H	1840	MSR_UNC_CBO_3_PERFEVTSEL0	Package	Uncore C-Box 3, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
736H	1846	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, Performance Counter 0
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
See Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29 for other MSR definitions applicable to processors with CPUID signatures 063CH, 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 CPUID DisplayFamily_DisplayModel signature 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 DisplayFamily_DisplayModel Signature 06_45H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	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.
		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)

Table 2-31. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with DisplayFamily_DisplayModel Signature 06_45H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
630H	1584	MSR_PKG_C8_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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
631H	1585	MSR_PKG_C9_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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
632H	1586	MSR_PKG_C10_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		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, Table 2-30 for other MSR definitions applicable to processors with CPUID signature 06_45H.				

2.14 MSRS IN INTEL® XEON® PROCESSOR E5 V3 AND E7 V3 PRODUCT FAMILY

Intel® Xeon® processor E5 v3 family and Intel® Xeon® processor E7 v3 family are based on Haswell-E microarchitecture (CPUID DisplayFamily_DisplayModel = 06_3F). These processors supports the MSR interfaces listed in Table 2-20, Table 2-29, and Table 2-32.

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
35H	53	MSR_CORE_THREAD_COUNT	Package	Configured State of Enabled Processor Core Count and Logical Processor Count (RO) ▪ 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 (RO) The number of logical processors that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.
		31:16		Core_COUNT (RO) The number of processor cores that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.
		63:32		Reserved
53H	83	MSR_THREAD_ID_INFO	Thread	A Hardware Assigned ID for the Logical Processor (RO)
		7:0		Logical_Processor_ID (RO) 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
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	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.
		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.

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		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
17DH	381	MSR_SMM_MCA_CAP	Thread	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		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
17FH	383	MSR_ERROR_CONTROL	Package	MC Bank Error Configuration (R/W)
		0		Reserved

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1		MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.
		63:2		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.
		55:48	Package	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.
		63:56	Package	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.
		15:8	Package	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.
		23:16	Package	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.
		31:24	Package	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.
		39:32	Package	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.
		47:40	Package	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.
		55:48	Package	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:56	Package	Maximum Ratio Limit for16C Maximum turbo ratio limit of 16 core active.
1AFH	431	MSR_TURBO_RATIO_LIMIT2	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 17C Maximum turbo ratio limit of 17 core active.
		15:8	Package	Maximum Ratio Limit for 18C Maximum turbo ratio limit of 18 core active.
		62:16	Package	Reserved
		63	Package	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).
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC5 reports MC errors from the Intel QPI 0 module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC7 reports MC errors from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC8 reports MC errors from the home agent HA 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "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.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "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.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "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.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC20 reports MC errors from the Intel QPI 1 module.
451H	1105	IA32_MC20_STATUS	Package	
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC21 reports MC errors from the Intel QPI 2 module.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 14.10.1, "RAPL Interfaces."
		7:4	Package	Reserved
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 14.10.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) Energy Consumed by DRAM devices.
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
61EH	1566	MSR_PCIE_PLL_RATIO	Package	Configuration of PCIE PLL Relative to BCLK(R/W)

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1:0	Package	PCIE Ratio (R/W) 00b: Use 5:5 mapping for100MHz operation (default). 01b: Use 5:4 mapping for125MHz operation. 10b: Use 5:3 mapping for166MHz operation. 11b: Use 5:2 mapping for250MHz operation.
		2	Package	LPLL Select (R/W) If 1, use configured setting of PCIE Ratio.
		3	Package	LONG RESET (R/W) If 1, wait an additional time-out before re-locking Gen2/Gen3 PLLs.
		63:4		Reserved
620H	1568	MSR UNCORE_RATIO_LIMIT	Package	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.
		63:15		Reserved
		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.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	Reserved (R/O) Reads return 0.
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		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 Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
		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.

Table 2-32. Additional MSRs Supported by Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
C8DH	3213	IA32_QM_EVTSEL	THREAD	Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.
		7:0		EventID (RW) Event encoding: 0x0: No monitoring. 0x1: L3 occupancy monitoring. All other encoding reserved.
		31:8		Reserved
		41:32		RMID (RW)
		63:42		Reserved
C8EH	3214	IA32_QM_CTR	THREAD	Monitoring Counter Register (R/O) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 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 and unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.
C8FH	3215	IA32_PQR_ASSOC	THREAD	Resource Association Register (R/W)
		9:0		RMID
		63: 10		Reserved
See Table 2-20, Table 2-29 for other MSR definitions applicable to processors with CPUID signature 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

Intel Xeon Processor E5 v3 and E7 v3 family are based on the Haswell-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-33. For complete detail of the uncore PMU, refer to Intel Xeon Processor E5 v3 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_3FH.

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
700H	1792	MSR_PMON_GLOBAL_CTL	Package	Uncore Perfmon Per-Socket Global Control
701H	1793	MSR_PMON_GLOBAL_STATUS	Package	Uncore Perfmon Per-Socket Global Status
702H	1794	MSR_PMON_GLOBAL_CONFIG	Package	Uncore Perfmon Per-Socket Global Configuration
703H	1795	MSR_U_PMON_UCLK_FIXED_CTL	Package	Uncore U-Box UCLK Fixed Counter Control
704H	1796	MSR_U_PMON_UCLK_FIXED_CTR	Package	Uncore U-Box UCLK Fixed Counter
705H	1797	MSR_U_PMON_EVNTSEL0	Package	Uncore U-Box Perfmon Event Select for U-Box Counter 0
706H	1798	MSR_U_PMON_EVNTSEL1	Package	Uncore U-Box Perfmon Event Select for U-Box Counter 1
708H	1800	MSR_U_PMON_BOX_STATUS	Package	Uncore U-Box Perfmon U-Box Wide Status
709H	1801	MSR_U_PMON_CTR0	Package	Uncore U-Box Perfmon Counter 0
70AH	1802	MSR_U_PMON_CTR1	Package	Uncore U-Box Perfmon Counter 1
710H	1808	MSR_PCU_PMON_BOX_CTL	Package	Uncore PCU Perfmon for PCU-Box-Wide Control
711H	1809	MSR_PCU_PMON_EVNTSEL0	Package	Uncore PCU Perfmon Event Select for PCU Counter 0
712H	1810	MSR_PCU_PMON_EVNTSEL1	Package	Uncore PCU Perfmon Event Select for PCU Counter 1
713H	1811	MSR_PCU_PMON_EVNTSEL2	Package	Uncore PCU Perfmon Event Select for PCU Counter 2
714H	1812	MSR_PCU_PMON_EVNTSEL3	Package	Uncore PCU Perfmon Event Select for PCU Counter 3
715H	1813	MSR_PCU_PMON_BOX_FILTER	Package	Uncore PCU Perfmon Box-Wide Filter
716H	1814	MSR_PCU_PMON_BOX_STATUS	Package	Uncore PCU Perfmon Box Wide Status
717H	1815	MSR_PCU_PMON_CTR0	Package	Uncore PCU Perfmon Counter 0
718H	1816	MSR_PCU_PMON_CTR1	Package	Uncore PCU Perfmon Counter 1
719H	1817	MSR_PCU_PMON_CTR2	Package	Uncore PCU Perfmon Counter 2
71AH	1818	MSR_PCU_PMON_CTR3	Package	Uncore PCU Perfmon Counter 3
720H	1824	MSR_S0_PMON_BOX_CTL	Package	Uncore SBo 0 Perfmon for SBo 0 Box-Wide Control
721H	1825	MSR_S0_PMON_EVNTSEL0	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 0
722H	1826	MSR_S0_PMON_EVNTSEL1	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 1
723H	1827	MSR_S0_PMON_EVNTSEL2	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 2
724H	1828	MSR_S0_PMON_EVNTSEL3	Package	Uncore SBo 0 Perfmon Event Select for SBo 0 Counter 3
725H	1829	MSR_S0_PMON_BOX_FILTER	Package	Uncore SBo 0 Perfmon Box-Wide Filter
726H	1830	MSR_S0_PMON_CTR0	Package	Uncore SBo 0 Perfmon Counter 0
727H	1831	MSR_S0_PMON_CTR1	Package	Uncore SBo 0 Perfmon Counter 1
728H	1832	MSR_S0_PMON_CTR2	Package	Uncore SBo 0 Perfmon Counter 2
729H	1833	MSR_S0_PMON_CTR3	Package	Uncore SBo 0 Perfmon Counter 3
72AH	1834	MSR_S1_PMON_BOX_CTL	Package	Uncore SBo 1 Perfmon for SBo 1 Box-Wide Control

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
72BH	1835	MSR_S1_PMON_EVNTSEL0	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 0
72CH	1836	MSR_S1_PMON_EVNTSEL1	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 1
72DH	1837	MSR_S1_PMON_EVNTSEL2	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 2
72EH	1838	MSR_S1_PMON_EVNTSEL3	Package	Uncore SBo 1 Perfmon Event Select for SBo 1 Counter 3
72FH	1839	MSR_S1_PMON_BOX_FILTER	Package	Uncore SBo 1 Perfmon Box-Wide Filter
730H	1840	MSR_S1_PMON_CTR0	Package	Uncore SBo 1 Perfmon Counter 0
731H	1841	MSR_S1_PMON_CTR1	Package	Uncore SBo 1 Perfmon Counter 1
732H	1842	MSR_S1_PMON_CTR2	Package	Uncore SBo 1 Perfmon Counter 2
733H	1843	MSR_S1_PMON_CTR3	Package	Uncore SBo 1 Perfmon Counter 3
734H	1844	MSR_S2_PMON_BOX_CTL	Package	Uncore SBo 2 Perfmon for SBo 2 Box-Wide Control
735H	1845	MSR_S2_PMON_EVNTSEL0	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 0
736H	1846	MSR_S2_PMON_EVNTSEL1	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 1
737H	1847	MSR_S2_PMON_EVNTSEL2	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 2
738H	1848	MSR_S2_PMON_EVNTSEL3	Package	Uncore SBo 2 Perfmon Event Select for SBo 2 Counter 3
739H	1849	MSR_S2_PMON_BOX_FILTER	Package	Uncore SBo 2 Perfmon Box-Wide Filter
73AH	1850	MSR_S2_PMON_CTR0	Package	Uncore SBo 2 Perfmon Counter 0
73BH	1851	MSR_S2_PMON_CTR1	Package	Uncore SBo 2 Perfmon Counter 1
73CH	1852	MSR_S2_PMON_CTR2	Package	Uncore SBo 2 Perfmon Counter 2
73DH	1853	MSR_S2_PMON_CTR3	Package	Uncore SBo 2 Perfmon Counter 3
73EH	1854	MSR_S3_PMON_BOX_CTL	Package	Uncore SBo 3 Perfmon for SBo 3 Box-Wide Control
73FH	1855	MSR_S3_PMON_EVNTSEL0	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 0
740H	1856	MSR_S3_PMON_EVNTSEL1	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 1
741H	1857	MSR_S3_PMON_EVNTSEL2	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 2
742H	1858	MSR_S3_PMON_EVNTSEL3	Package	Uncore SBo 3 Perfmon Event Select for SBo 3 Counter 3
743H	1859	MSR_S3_PMON_BOX_FILTER	Package	Uncore SBo 3 Perfmon Box-Wide Filter
744H	1860	MSR_S3_PMON_CTR0	Package	Uncore SBo 3 Perfmon Counter 0
745H	1861	MSR_S3_PMON_CTR1	Package	Uncore SBo 3 Perfmon Counter 1
746H	1862	MSR_S3_PMON_CTR2	Package	Uncore SBo 3 Perfmon Counter 2
747H	1863	MSR_S3_PMON_CTR3	Package	Uncore SBo 3 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E00H	3584	MSR_C0_PMON_BOX_CTL	Package	Uncore C-Box 0 Perfmon for Box-Wide Control
E01H	3585	MSR_C0_PMON_EVNTSEL0	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 0
E02H	3586	MSR_C0_PMON_EVNTSEL1	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 1
E03H	3587	MSR_C0_PMON_EVNTSEL2	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 2
E04H	3588	MSR_C0_PMON_EVNTSEL3	Package	Uncore C-Box 0 Perfmon Event Select for C-Box 0 Counter 3
E05H	3589	MSR_C0_PMON_BOX_FILTER0	Package	Uncore C-Box 0 Perfmon Box Wide Filter 0
E06H	3590	MSR_C0_PMON_BOX_FILTER1	Package	Uncore C-Box 0 Perfmon Box Wide Filter 1
E07H	3591	MSR_C0_PMON_BOX_STATUS	Package	Uncore C-Box 0 Perfmon Box Wide Status
E08H	3592	MSR_C0_PMON_CTR0	Package	Uncore C-Box 0 Perfmon Counter 0
E09H	3593	MSR_C0_PMON_CTR1	Package	Uncore C-Box 0 Perfmon Counter 1
E0AH	3594	MSR_C0_PMON_CTR2	Package	Uncore C-Box 0 Perfmon Counter 2
E0BH	3595	MSR_C0_PMON_CTR3	Package	Uncore C-Box 0 Perfmon Counter 3
E10H	3600	MSR_C1_PMON_BOX_CTL	Package	Uncore C-Box 1 Perfmon for Box-Wide Control
E11H	3601	MSR_C1_PMON_EVNTSEL0	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 0
E12H	3602	MSR_C1_PMON_EVNTSEL1	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 1
E13H	3603	MSR_C1_PMON_EVNTSEL2	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 2
E14H	3604	MSR_C1_PMON_EVNTSEL3	Package	Uncore C-Box 1 Perfmon Event Select for C-Box 1 Counter 3
E15H	3605	MSR_C1_PMON_BOX_FILTER0	Package	Uncore C-Box 1 Perfmon Box Wide Filter 0
E16H	3606	MSR_C1_PMON_BOX_FILTER1	Package	Uncore C-Box 1 Perfmon Box Wide Filter1
E17H	3607	MSR_C1_PMON_BOX_STATUS	Package	Uncore C-Box 1 Perfmon Box Wide Status
E18H	3608	MSR_C1_PMON_CTR0	Package	Uncore C-Box 1 Perfmon Counter 0
E19H	3609	MSR_C1_PMON_CTR1	Package	Uncore C-Box 1 Perfmon Counter 1
E1AH	3610	MSR_C1_PMON_CTR2	Package	Uncore C-Box 1 Perfmon Counter 2
E1BH	3611	MSR_C1_PMON_CTR3	Package	Uncore C-Box 1 Perfmon Counter 3
E20H	3616	MSR_C2_PMON_BOX_CTL	Package	Uncore C-Box 2 Perfmon for Box-Wide Control
E21H	3617	MSR_C2_PMON_EVNTSEL0	Package	Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 0
E22H	3618	MSR_C2_PMON_EVNTSEL1	Package	Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 1
E23H	3619	MSR_C2_PMON_EVNTSEL2	Package	Uncore C-Box 2 Perfmon Event Select for C-Box 2 Counter 2
E24H	3620	MSR_C2_PMON_EVNTSEL3	Package	Uncore C-Box 2 Perfmon Event select for C-Box 2 Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E25H	3621	MSR_C2_PMON_BOX_FILTER0	Package	Uncore C-Box 2 Perfmon Box Wide Filter 0
E26H	3622	MSR_C2_PMON_BOX_FILTER1	Package	Uncore C-Box 2 Perfmon Box Wide Filter1
E27H	3623	MSR_C2_PMON_BOX_STATUS	Package	Uncore C-Box 2 Perfmon Box Wide Status
E28H	3624	MSR_C2_PMON_CTR0	Package	Uncore C-Box 2 Perfmon Counter 0
E29H	3625	MSR_C2_PMON_CTR1	Package	Uncore C-Box 2 Perfmon Counter 1
E2AH	3626	MSR_C2_PMON_CTR2	Package	Uncore C-Box 2 Perfmon Counter 2
E2BH	3627	MSR_C2_PMON_CTR3	Package	Uncore C-Box 2 Perfmon Counter 3
E30H	3632	MSR_C3_PMON_BOX_CTL	Package	Uncore C-Box 3 Perfmon for Box-Wide Control
E31H	3633	MSR_C3_PMON_EVNTSEL0	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 0
E32H	3634	MSR_C3_PMON_EVNTSEL1	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 1
E33H	3635	MSR_C3_PMON_EVNTSEL2	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 2
E34H	3636	MSR_C3_PMON_EVNTSEL3	Package	Uncore C-Box 3 Perfmon Event Select for C-Box 3 Counter 3
E35H	3637	MSR_C3_PMON_BOX_FILTER0	Package	Uncore C-Box 3 Perfmon Box Wide Filter 0
E36H	3638	MSR_C3_PMON_BOX_FILTER1	Package	Uncore C-Box 3 Perfmon Box Wide Filter1
E37H	3639	MSR_C3_PMON_BOX_STATUS	Package	Uncore C-Box 3 Perfmon Box Wide Status
E38H	3640	MSR_C3_PMON_CTR0	Package	Uncore C-Box 3 Perfmon Counter 0
E39H	3641	MSR_C3_PMON_CTR1	Package	Uncore C-Box 3 Perfmon Counter 1
E3AH	3642	MSR_C3_PMON_CTR2	Package	Uncore C-Box 3 Perfmon Counter 2
E3BH	3643	MSR_C3_PMON_CTR3	Package	Uncore C-Box 3 Perfmon Counter 3
E40H	3648	MSR_C4_PMON_BOX_CTL	Package	Uncore C-Box 4 Perfmon for Box-Wide Control
E41H	3649	MSR_C4_PMON_EVNTSEL0	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 0
E42H	3650	MSR_C4_PMON_EVNTSEL1	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 1
E43H	3651	MSR_C4_PMON_EVNTSEL2	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 2
E44H	3652	MSR_C4_PMON_EVNTSEL3	Package	Uncore C-Box 4 Perfmon Event Select for C-Box 4 Counter 3
E45H	3653	MSR_C4_PMON_BOX_FILTER0	Package	Uncore C-Box 4 Perfmon Box Wide Filter 0
E46H	3654	MSR_C4_PMON_BOX_FILTER1	Package	Uncore C-Box 4 Perfmon Box Wide Filter1
E47H	3655	MSR_C4_PMON_BOX_STATUS	Package	Uncore C-Box 4 Perfmon Box Wide Status
E48H	3656	MSR_C4_PMON_CTR0	Package	Uncore C-Box 4 Perfmon Counter 0
E49H	3657	MSR_C4_PMON_CTR1	Package	Uncore C-Box 4 Perfmon Counter 1
E4AH	3658	MSR_C4_PMON_CTR2	Package	Uncore C-Box 4 Perfmon Counter 2
E4BH	3659	MSR_C4_PMON_CTR3	Package	Uncore C-Box 4 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E50H	3664	MSR_C5_PMON_BOX_CTL	Package	Uncore C-Box 5 Perfmon for Box-Wide Control
E51H	3665	MSR_C5_PMON_EVNTSEL0	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 0
E52H	3666	MSR_C5_PMON_EVNTSEL1	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 1
E53H	3667	MSR_C5_PMON_EVNTSEL2	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 2
E54H	3668	MSR_C5_PMON_EVNTSEL3	Package	Uncore C-Box 5 Perfmon Event Select for C-Box 5 Counter 3
E55H	3669	MSR_C5_PMON_BOX_FILTER0	Package	Uncore C-Box 5 Perfmon Box Wide Filter 0
E56H	3670	MSR_C5_PMON_BOX_FILTER1	Package	Uncore C-Box 5 Perfmon Box Wide Filter 1
E57H	3671	MSR_C5_PMON_BOX_STATUS	Package	Uncore C-Box 5 Perfmon Box Wide Status
E58H	3672	MSR_C5_PMON_CTR0	Package	Uncore C-Box 5 Perfmon Counter 0
E59H	3673	MSR_C5_PMON_CTR1	Package	Uncore C-Box 5 Perfmon Counter 1
E5AH	3674	MSR_C5_PMON_CTR2	Package	Uncore C-Box 5 Perfmon Counter 2
E5BH	3675	MSR_C5_PMON_CTR3	Package	Uncore C-Box 5 Perfmon Counter 3
E60H	3680	MSR_C6_PMON_BOX_CTL	Package	Uncore C-Box 6 Perfmon for Box-Wide Control
E61H	3681	MSR_C6_PMON_EVNTSEL0	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 0
E62H	3682	MSR_C6_PMON_EVNTSEL1	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 1
E63H	3683	MSR_C6_PMON_EVNTSEL2	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 2
E64H	3684	MSR_C6_PMON_EVNTSEL3	Package	Uncore C-Box 6 Perfmon Event Select for C-Box 6 Counter 3
E65H	3685	MSR_C6_PMON_BOX_FILTER0	Package	Uncore C-Box 6 Perfmon Box Wide Filter 0
E66H	3686	MSR_C6_PMON_BOX_FILTER1	Package	Uncore C-Box 6 Perfmon Box Wide Filter 1
E67H	3687	MSR_C6_PMON_BOX_STATUS	Package	Uncore C-Box 6 Perfmon Box Wide Status
E68H	3688	MSR_C6_PMON_CTR0	Package	Uncore C-Box 6 Perfmon Counter 0
E69H	3689	MSR_C6_PMON_CTR1	Package	Uncore C-Box 6 Perfmon Counter 1
E6AH	3690	MSR_C6_PMON_CTR2	Package	Uncore C-Box 6 Perfmon Counter 2
E6BH	3691	MSR_C6_PMON_CTR3	Package	Uncore C-Box 6 Perfmon Counter 3
E70H	3696	MSR_C7_PMON_BOX_CTL	Package	Uncore C-Box 7 Perfmon for Box-Wide Control
E71H	3697	MSR_C7_PMON_EVNTSEL0	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 0
E72H	3698	MSR_C7_PMON_EVNTSEL1	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 1
E73H	3699	MSR_C7_PMON_EVNTSEL2	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 2
E74H	3700	MSR_C7_PMON_EVNTSEL3	Package	Uncore C-Box 7 Perfmon Event Select for C-Box 7 Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E75H	3701	MSR_C7_PMON_BOX_FILTER0	Package	Uncore C-Box 7 Perfmon Box Wide Filter 0
E76H	3702	MSR_C7_PMON_BOX_FILTER1	Package	Uncore C-Box 7 Perfmon Box Wide Filter 1
E77H	3703	MSR_C7_PMON_BOX_STATUS	Package	Uncore C-Box 7 Perfmon Box Wide Status
E78H	3704	MSR_C7_PMON_CTR0	Package	Uncore C-Box 7 Perfmon Counter 0
E79H	3705	MSR_C7_PMON_CTR1	Package	Uncore C-Box 7 Perfmon Counter 1
E7AH	3706	MSR_C7_PMON_CTR2	Package	Uncore C-Box 7 Perfmon Counter 2
E7BH	3707	MSR_C7_PMON_CTR3	Package	Uncore C-Box 7 Perfmon Counter 3
E80H	3712	MSR_C8_PMON_BOX_CTL	Package	Uncore C-Box 8 Perfmon Local Box Wide Control
E81H	3713	MSR_C8_PMON_EVNTSEL0	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 0
E82H	3714	MSR_C8_PMON_EVNTSEL1	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 1
E83H	3715	MSR_C8_PMON_EVNTSEL2	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 2
E84H	3716	MSR_C8_PMON_EVNTSEL3	Package	Uncore C-Box 8 Perfmon Event Select for C-Box 8 Counter 3
E85H	3717	MSR_C8_PMON_BOX_FILTER0	Package	Uncore C-Box 8 Perfmon Box Wide Filter 0
E86H	3718	MSR_C8_PMON_BOX_FILTER1	Package	Uncore C-Box 8 Perfmon Box Wide Filter 1
E87H	3719	MSR_C8_PMON_BOX_STATUS	Package	Uncore C-Box 8 Perfmon Box Wide Status
E88H	3720	MSR_C8_PMON_CTR0	Package	Uncore C-Box 8 Perfmon Counter 0
E89H	3721	MSR_C8_PMON_CTR1	Package	Uncore C-Box 8 Perfmon Counter 1
E8AH	3722	MSR_C8_PMON_CTR2	Package	Uncore C-Box 8 Perfmon Counter 2
E8BH	3723	MSR_C8_PMON_CTR3	Package	Uncore C-Box 8 Perfmon Counter 3
E90H	3728	MSR_C9_PMON_BOX_CTL	Package	Uncore C-Box 9 Perfmon Local Box Wide Control
E91H	3729	MSR_C9_PMON_EVNTSEL0	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 0
E92H	3730	MSR_C9_PMON_EVNTSEL1	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 1
E93H	3731	MSR_C9_PMON_EVNTSEL2	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 2
E94H	3732	MSR_C9_PMON_EVNTSEL3	Package	Uncore C-Box 9 Perfmon Event Select for C-Box 9 Counter 3
E95H	3733	MSR_C9_PMON_BOX_FILTER0	Package	Uncore C-Box 9 Perfmon Box Wide Filter 0
E96H	3734	MSR_C9_PMON_BOX_FILTER1	Package	Uncore C-Box 9 Perfmon Box Wide Filter 1
E97H	3735	MSR_C9_PMON_BOX_STATUS	Package	Uncore C-Box 9 Perfmon Box Wide Status
E98H	3736	MSR_C9_PMON_CTR0	Package	Uncore C-Box 9 Perfmon Counter 0
E99H	3737	MSR_C9_PMON_CTR1	Package	Uncore C-Box 9 Perfmon Counter 1
E9AH	3738	MSR_C9_PMON_CTR2	Package	Uncore C-Box 9 Perfmon Counter 2
E9BH	3739	MSR_C9_PMON_CTR3	Package	Uncore C-Box 9 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
EA0H	3744	MSR_C10_PMON_BOX_CTL	Package	Uncore C-Box 10 Perfmon Local Box Wide Control
EA1H	3745	MSR_C10_PMON_EVNTSEL0	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 0
EA2H	3746	MSR_C10_PMON_EVNTSEL1	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 1
EA3H	3747	MSR_C10_PMON_EVNTSEL2	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 2
EA4H	3748	MSR_C10_PMON_EVNTSEL3	Package	Uncore C-Box 10 Perfmon Event Select for C-Box 10 Counter 3
EA5H	3749	MSR_C10_PMON_BOX_FILTER0	Package	Uncore C-Box 10 Perfmon Box Wide Filter 0
EA6H	3750	MSR_C10_PMON_BOX_FILTER1	Package	Uncore C-Box 10 Perfmon Box Wide Filter 1
EA7H	3751	MSR_C10_PMON_BOX_STATUS	Package	Uncore C-Box 10 Perfmon Box Wide Status
EA8H	3752	MSR_C10_PMON_CTR0	Package	Uncore C-Box 10 Perfmon Counter 0
EA9H	3753	MSR_C10_PMON_CTR1	Package	Uncore C-Box 10 perfmon Counter 1
EAAH	3754	MSR_C10_PMON_CTR2	Package	Uncore C-Box 10 Perfmon Counter 2
EABH	3755	MSR_C10_PMON_CTR3	Package	Uncore C-Box 10 Perfmon Counter 3
EB0H	3760	MSR_C11_PMON_BOX_CTL	Package	Uncore C-Box 11 Perfmon Local Box Wide Control
EB1H	3761	MSR_C11_PMON_EVNTSEL0	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 0
EB2H	3762	MSR_C11_PMON_EVNTSEL1	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 1
EB3H	3763	MSR_C11_PMON_EVNTSEL2	Package	Uncore C-Box 11 Perfmon Event Select for C-Box 11 Counter 2
EB4H	3764	MSR_C11_PMON_EVNTSEL3	Package	Uncore C-box 11 Perfmon Event Select for C-Box 11 Counter 3
EB5H	3765	MSR_C11_PMON_BOX_FILTER0	Package	Uncore C-Box 11 Perfmon Box Wide Filter 0
EB6H	3766	MSR_C11_PMON_BOX_FILTER1	Package	Uncore C-Box 11 Perfmon Box Wide Filter 1
EB7H	3767	MSR_C11_PMON_BOX_STATUS	Package	Uncore C-Box 11 Perfmon Box Wide Status
EB8H	3768	MSR_C11_PMON_CTR0	Package	Uncore C-Box 11 Perfmon Counter 0
EB9H	3769	MSR_C11_PMON_CTR1	Package	Uncore C-Box 11 Perfmon Counter 1
EBAH	3770	MSR_C11_PMON_CTR2	Package	Uncore C-Box 11 Perfmon Counter 2
EBBH	3771	MSR_C11_PMON_CTR3	Package	Uncore C-Box 11 Perfmon Counter 3
EC0H	3776	MSR_C12_PMON_BOX_CTL	Package	Uncore C-Box 12 Perfmon Local Box Wide Control
EC1H	3777	MSR_C12_PMON_EVNTSEL0	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 0
EC2H	3778	MSR_C12_PMON_EVNTSEL1	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 1
EC3H	3779	MSR_C12_PMON_EVNTSEL2	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 2
EC4H	3780	MSR_C12_PMON_EVNTSEL3	Package	Uncore C-Box 12 Perfmon Event Select for C-Box 12 Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
EC5H	3781	MSR_C12_PMON_BOX_FILTER0	Package	Uncore C-Box 12 Perfmon Box Wide Filter 0
EC6H	3782	MSR_C12_PMON_BOX_FILTER1	Package	Uncore C-Box 12 Perfmon Box Wide Filter 1
EC7H	3783	MSR_C12_PMON_BOX_STATUS	Package	Uncore C-Box 12 Perfmon Box Wide Status
EC8H	3784	MSR_C12_PMON_CTR0	Package	Uncore C-Box 12 Perfmon Counter 0
EC9H	3785	MSR_C12_PMON_CTR1	Package	Uncore C-Box 12 Perfmon Counter 1
ECAH	3786	MSR_C12_PMON_CTR2	Package	Uncore C-Box 12 Perfmon Counter 2
ECBH	3787	MSR_C12_PMON_CTR3	Package	Uncore C-Box 12 Perfmon Counter 3
ED0H	3792	MSR_C13_PMON_BOX_CTL	Package	Uncore C-Box 13 Perfmon local box wide control.
ED1H	3793	MSR_C13_PMON_EVNTSEL0	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 0
ED2H	3794	MSR_C13_PMON_EVNTSEL1	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 1
ED3H	3795	MSR_C13_PMON_EVNTSEL2	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 2
ED4H	3796	MSR_C13_PMON_EVNTSEL3	Package	Uncore C-Box 13 Perfmon Event Select for C-Box 13 Counter 3
ED5H	3797	MSR_C13_PMON_BOX_FILTER0	Package	Uncore C-Box 13 Perfmon Box Wide Filter 0
ED6H	3798	MSR_C13_PMON_BOX_FILTER1	Package	Uncore C-Box 13 Perfmon Box Wide Filter 1
ED7H	3799	MSR_C13_PMON_BOX_STATUS	Package	Uncore C-Box 13 Perfmon Box Wide Status
ED8H	3800	MSR_C13_PMON_CTR0	Package	Uncore C-Box 13 Perfmon Counter 0
ED9H	3801	MSR_C13_PMON_CTR1	Package	Uncore C-Box 13 Perfmon Counter 1
EDAH	3802	MSR_C13_PMON_CTR2	Package	Uncore C-Box 13 Perfmon Counter 2
EDBH	3803	MSR_C13_PMON_CTR3	Package	Uncore C-Box 13 Perfmon Counter 3
EE0H	3808	MSR_C14_PMON_BOX_CTL	Package	Uncore C-Box 14 Perfmon Local Box Wide Control
EE1H	3809	MSR_C14_PMON_EVNTSEL0	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 0
EE2H	3810	MSR_C14_PMON_EVNTSEL1	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 1
EE3H	3811	MSR_C14_PMON_EVNTSEL2	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 2
EE4H	3812	MSR_C14_PMON_EVNTSEL3	Package	Uncore C-Box 14 Perfmon Event Select for C-Box 14 Counter 3
EE5H	3813	MSR_C14_PMON_BOX_FILTER	Package	Uncore C-Box 14 Perfmon Box Wide Filter 0
EE6H	3814	MSR_C14_PMON_BOX_FILTER1	Package	Uncore C-Box 14 Perfmon Box Wide Filter 1
EE7H	3815	MSR_C14_PMON_BOX_STATUS	Package	Uncore C-Box 14 Perfmon Box Wide Status
EE8H	3816	MSR_C14_PMON_CTR0	Package	Uncore C-Box 14 Perfmon Counter 0
EE9H	3817	MSR_C14_PMON_CTR1	Package	Uncore C-Box 14 Perfmon Counter 1
EEAH	3818	MSR_C14_PMON_CTR2	Package	Uncore C-Box 14 Perfmon Counter 2
EEBH	3819	MSR_C14_PMON_CTR3	Package	Uncore C-Box 14 Perfmon Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
EF0H	3824	MSR_C15_PMON_BOX_CTL	Package	Uncore C-Box 15 Perfmon Local Box Wide Control
EF1H	3825	MSR_C15_PMON_EVNTSEL0	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 0
EF2H	3826	MSR_C15_PMON_EVNTSEL1	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 1
EF3H	3827	MSR_C15_PMON_EVNTSEL2	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 2
EF4H	3828	MSR_C15_PMON_EVNTSEL3	Package	Uncore C-Box 15 Perfmon Event Select for C-Box 15 Counter 3
EF5H	3829	MSR_C15_PMON_BOX_FILTER0	Package	Uncore C-Box 15 Perfmon Box Wide Filter 0
EF6H	3830	MSR_C15_PMON_BOX_FILTER1	Package	Uncore C-Box 15 Perfmon Box Wide Filter 1
EF7H	3831	MSR_C15_PMON_BOX_STATUS	Package	Uncore C-Box 15 Perfmon Box Wide Status
EF8H	3832	MSR_C15_PMON_CTR0	Package	Uncore C-Box 15 Perfmon Counter 0
EF9H	3833	MSR_C15_PMON_CTR1	Package	Uncore C-Box 15 Perfmon Counter 1
EFAH	3834	MSR_C15_PMON_CTR2	Package	Uncore C-Box 15 Perfmon Counter 2
EFBH	3835	MSR_C15_PMON_CTR3	Package	Uncore C-Box 15 Perfmon Counter 3
F00H	3840	MSR_C16_PMON_BOX_CTL	Package	Uncore C-Box 16 Perfmon for Box-Wide Control
F01H	3841	MSR_C16_PMON_EVNTSEL0	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 0
F02H	3842	MSR_C16_PMON_EVNTSEL1	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 1
F03H	3843	MSR_C16_PMON_EVNTSEL2	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 2
F04H	3844	MSR_C16_PMON_EVNTSEL3	Package	Uncore C-Box 16 Perfmon Event Select for C-Box 16 Counter 3
F05H	3845	MSR_C16_PMON_BOX_FILTER0	Package	Uncore C-Box 16 Perfmon Box Wide Filter 0
F06H	3846	MSR_C16_PMON_BOX_FILTER1	Package	Uncore C-Box 16 Perfmon Box Wide Filter 1
F07H	3847	MSR_C16_PMON_BOX_STATUS	Package	Uncore C-Box 16 Perfmon Box Wide Status
F08H	3848	MSR_C16_PMON_CTR0	Package	Uncore C-Box 16 Perfmon Counter 0
F09H	3849	MSR_C16_PMON_CTR1	Package	Uncore C-Box 16 Perfmon Counter 1
F0AH	3850	MSR_C16_PMON_CTR2	Package	Uncore C-Box 16 Perfmon Counter 2
F0BH	3851	MSR_C16_PMON_CTR3	Package	Uncore C-Box 16 Perfmon Counter 3
F10H	3856	MSR_C17_PMON_BOX_CTL	Package	Uncore C-Box 17 Perfmon for Box-Wide Control
F11H	3857	MSR_C17_PMON_EVNTSEL0	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 0
F12H	3858	MSR_C17_PMON_EVNTSEL1	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 1
F13H	3859	MSR_C17_PMON_EVNTSEL2	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 2
F14H	3860	MSR_C17_PMON_EVNTSEL3	Package	Uncore C-Box 17 Perfmon Event Select for C-Box 17 Counter 3

Table 2-33. Uncore PMU MSRs in Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
F15H	3861	MSR_C17_PMON_BOX_FILTER0	Package	Uncore C-Box 17 Perfmon Box Wide Filter 0
F16H	3862	MSR_C17_PMON_BOX_FILTER1	Package	Uncore C-Box 17 Perfmon Box Wide Filter1
F17H	3863	MSR_C17_PMON_BOX_STATUS	Package	Uncore C-Box 17 Perfmon Box Wide Status
F18H	3864	MSR_C17_PMON_CTR0	Package	Uncore C-Box 17 Perfmon Counter 0
F19H	3865	MSR_C17_PMON_CTR1	Package	Uncore C-Box 17 Perfmon Counter 1
F1AH	3866	MSR_C17_PMON_CTR2	Package	Uncore C-Box 17 Perfmon Counter 2
F1BH	3867	MSR_C17_PMON_CTR3	Package	Uncore C-Box 17 Perfmon Counter 3

2.15 MSRS IN INTEL® CORE™ M PROCESSORS AND 5TH GENERATION INTEL CORE PROCESSORS

The Intel® Core™ M-5xxx processors and 5th generation Intel® Core™ Processors, and Intel® Xeon® Processor E3-1200 v4 family are based on the Broadwell microarchitecture. The Intel® Core™ M-5xxx processors and 5th generation Intel® Core™ Processors have CPUID DisplayFamily_DisplayModel signature 06_3DH. Intel® Xeon® Processor E3-1200 v4 family and the 5th generation Intel® Core™ Processors have CPUID DisplayFamily_DisplayModel signature 06_47H. Processors with signatures 06_3DH and 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 supersede prior generation tables.

Table 2-34 lists MSRs that are common to processors based on the Broadwell microarchitectures (including CPUID signatures 06_3DH, 06_47H, 06_4FH, and 06_56H).

Table 2-34. Additional MSRs Common to Processors Based the Broadwell Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
		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 32.2.7.2, "Table of Physical Addresses (ToPA)."
		60:56		Reserved

Table 2-34. Additional MSRs Common to Processors Based the Broadwell Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		61		Ovf_Uncore
		62		Ovf_BufDSSAVE
		63		CondChgd
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 2-2. See Section 19.6.2.2, "Global Counter Control Facilities."
		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 32.2.7.2, "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
560H	1376	IA32_RTIT_OUTPUT_BASE	THREAD	Trace Output Base Register (R/W)
		6:0		Reserved
		MAXPHYADDR[1]-1:7		Base physical address.
		63:MAXPHYADDR		Reserved
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	THREAD	Trace Output Mask Pointers Register (R/W)
		6:0		Reserved
		31:7		MaskOrTableOffset
		63:32		Output Offset.
570H	1392	IA32_RTIT_CTL	Thread	Trace Control Register (R/W)
		0		TraceEn
		1		Reserved, must be zero.
		2		OS
		3		User
		6:4		Reserved, must be zero.
		7		CR3 filter
		8		ToPA Writing 0 will #GP if also setting TraceEn.

Table 2-34. Additional MSRs Common to Processors Based the Broadwell Microarchitectures

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
571H	1393	IA32_RTIT_STATUS	Thread	Tracing Status Register (R/W)
		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.
572H	1394	IA32_RTIT_CR3_MATCH	THREAD	Trace Filter CR3 Match Register (R/W)
		4:0		Reserved
		63:5		CR3[63:5] value to match.
620H	1568	MSR UNCORE_RATIO_LIMIT	Package	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.
		63:15		Reserved
		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.

NOTES:

1. MAXPHYADDR is reported by CPUID.80000008H:EAX[7:0].

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		Register Name	Scope	Bit Description
Hex	Dec			
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	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.
		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)
		29		Enable Package C-State Auto-Demotion (R/W)
		30		Enable Package C-State Undemotion (R/W)
		63:31		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		39:32	Package	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5core active.
		47:40	Package	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6core active.
		63:48		Reserved
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
See Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29, Table 2-30, Table 2-34 for other MSR definitions applicable to processors with CPUID signature 06_3DH.				

2.16 MSRS IN INTEL® XEON® PROCESSORS E5 V4 FAMILY

The MSRs listed in Table 2-36 are available and common to Intel® Xeon® Processor D product Family (CPUID DisplayFamily_DisplayModel = 06_56H) and to Intel Xeon processors E5 v4, E7 v4 families (CPUID DisplayFamily_DisplayModel = 06_4FH). They are based on the Broadwell microarchitecture.

See Section 2.16.1 for lists of tables of MSRs that are supported by Intel® Xeon® Processor D Family.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) See Table 2-26.
		22:16		Reserved.
		23	Package	PPIN_CAP (R/O) See Table 2-26.
		27:24		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.
		30	Package	Programmable TJ OFFSET (R/O) See Table 2-26.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) See Table 2-26.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	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.
		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)

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		16		Automatic C-State Conversion Enable (R/W) If 1, the processor will convert HALT or MWAT(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
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		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
17DH	381	MSR_SMM_MCA_CAP	Thread	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		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
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		0		Thermal Status (RO) See Table 2-2.
		1		Thermal Status Log (R/WC0) See Table 2-2.
		2		PROTCHOT # or FORCEPR# Status (RO) See Table 2-2.
		3		PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.
		4		Critical Temperature Status (RO) See Table 2-2.
		5		Critical Temperature Status Log (R/WC0) See Table 2-2.
		6		Thermal Threshold #1 Status (RO) See Table 2-2.
		7		Thermal Threshold #1 Log (R/WC0) See Table 2-2.
		8		Thermal Threshold #2 Status (RO) See Table 2-2.
		9		Thermal Threshold #2 Log (R/WC0) See Table 2-2.
		10		Power Limitation Status (RO) See Table 2-2.
		11		Power Limitation Log (R/WC0) See Table 2-2.
		12		Current Limit Status (RO) See Table 2-2.
		13		Current Limit Log (R/WC0) See Table 2-2.
		14		Cross Domain Limit Status (RO) See Table 2-2.
		15		Cross Domain Limit Log (R/WC0) See Table 2-2.
		22:16		Digital Readout (RO) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (RO) See Table 2-2.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31		Reading Valid (RO) See Table 2-2.
		63:32		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (RO) See Table 2-26.
		27:24		TCC Activation Offset (R/W) See Table 2-26.
		63:28		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 1C
		15:8	Package	Maximum Ratio Limit for 2C
		23:16	Package	Maximum Ratio Limit for 3C
		31:24	Package	Maximum Ratio Limit for 4C
		39:32	Package	Maximum Ratio Limit for 5C
		47:40	Package	Maximum Ratio Limit for 6C
		55:48	Package	Maximum Ratio Limit for 7C
		63:56	Package	Maximum Ratio Limit for 8C
1AEH	430	MSR_TURBO_RATIO_LIMIT1	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		7:0	Package	Maximum Ratio Limit for 9C
		15:8	Package	Maximum Ratio Limit for 10C
		23:16	Package	Maximum Ratio Limit for 11C
		31:24	Package	Maximum Ratio Limit for 12C
		39:32	Package	Maximum Ratio Limit for 13C
		47:40	Package	Maximum Ratio Limit for 14C
		55:48	Package	Maximum Ratio Limit for 15C
		63:56	Package	Maximum Ratio Limit for 16C
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 14.10.1, "RAPL Interfaces."
		7:4	Package	Reserved

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 14.10.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) Energy consumed by DRAM devices.
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
620H	1568	MSR UNCORE_RATIO_LIMIT	Package	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.
		63:15		Reserved
		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.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	Reserved (R/O) Reads return 0.
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
		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.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
770H	1904	IA32_PM_ENABLE	Package	See Section 14.4.2, "Enabling HWP".
771H	1905	IA32_HWP_CAPABILITIES	Thread	See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".
774H	1908	IA32_HWP_REQUEST	Thread	See Section 14.4.4, "Managing HWP".
		7:0		Minimum Performance (R/W)
		15:8		Maximum Performance (R/W)
		23:16		Desired Performance (R/W)
		63:24		Reserved
777H	1911	IA32_HWP_STATUS	Thread	See Section 14.4.5, "HWP Feedback".
		1:0		Reserved
		2		Excursion to Minimum (RO)
		63:3		Reserved
C8DH	3213	IA32_QM_EVTSEL	Thread	Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.
		7:0		EventID (RW) Event encoding: 0x00: No monitoring. 0x01: L3 occupancy monitoring. 0x02: Total memory bandwidth monitoring. 0x03: Local memory bandwidth monitoring. All other encoding reserved.

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		31:8		Reserved
		41:32		RMID (RW)
		63:42		Reserved
C8FH	3215	IA32_PQR_ASSOC	THREAD	Resource Association Register (R/W)
		9:0		RMID
		31:10		Reserved
		51:32		COS (R/W)
		63: 52		Reserved
C90H	3216	IA32_L3_QOS_MASK_0	Package	L3 Class Of Service Mask - COS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.
		0:19		CBM: Bit vector of available L3 ways for COS 0 enforcement.
		63:20		Reserved
C91H	3217	IA32_L3_QOS_MASK_1	Package	L3 Class Of Service Mask - COS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.
		0:19		CBM: Bit vector of available L3 ways for COS 1 enforcement.
		63:20		Reserved
C92H	3218	IA32_L3_QOS_MASK_2	Package	L3 Class Of Service Mask - COS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.
		0:19		CBM: Bit vector of available L3 ways for COS 2 enforcement.
		63:20		Reserved
C93H	3219	IA32_L3_QOS_MASK_3	Package	L3 Class Of Service Mask - COS 3 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.
		0:19		CBM: Bit vector of available L3 ways for COS 3 enforcement.
		63:20		Reserved
C94H	3220	IA32_L3_QOS_MASK_4	Package	L3 Class Of Service Mask - COS 4 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4.
		0:19		CBM: Bit vector of available L3 ways for COS 4 enforcement.
		63:20		Reserved
C95H	3221	IA32_L3_QOS_MASK_5	Package	L3 Class Of Service Mask - COS 5 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5.
		0:19		CBM: Bit vector of available L3 ways for COS 5 enforcement.
		63:20		Reserved

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C96H	3222	IA32_L3_QOS_MASK_6	Package	L3 Class Of Service Mask - COS 6 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=6.
		0:19		CBM: Bit vector of available L3 ways for COS 6 enforcement.
		63:20		Reserved
C97H	3223	IA32_L3_QOS_MASK_7	Package	L3 Class Of Service Mask - COS 7 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=7.
		0:19		CBM: Bit vector of available L3 ways for COS 7 enforcement.
		63:20		Reserved
C98H	3224	IA32_L3_QOS_MASK_8	Package	L3 Class Of Service Mask - COS 8 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=8.
		0:19		CBM: Bit vector of available L3 ways for COS 8 enforcement.
		63:20		Reserved
C99H	3225	IA32_L3_QOS_MASK_9	Package	L3 Class Of Service Mask - COS 9 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=9.
		0:19		CBM: Bit vector of available L3 ways for COS 9 enforcement.
		63:20		Reserved
C9AH	3226	IA32_L3_QOS_MASK_10	Package	L3 Class Of Service Mask - COS 10 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=10.
		0:19		CBM: Bit vector of available L3 ways for COS 10 enforcement.
		63:20		Reserved
C9BH	3227	IA32_L3_QOS_MASK_11	Package	L3 Class Of Service Mask - COS 11 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=11.
		0:19		CBM: Bit vector of available L3 ways for COS 11 enforcement.
		63:20		Reserved
C9CH	3228	IA32_L3_QOS_MASK_12	Package	L3 Class Of Service Mask - COS 12 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=12.
		0:19		CBM: Bit vector of available L3 ways for COS 12 enforcement.
		63:20		Reserved
C9DH	3229	IA32_L3_QOS_MASK_13	Package	L3 Class Of Service Mask - COS 13 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=13.
		0:19		CBM: Bit vector of available L3 ways for COS 13 enforcement.
		63:20		Reserved

Table 2-36. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
C9EH	3230	IA32_L3_QOS_MASK_14	Package	L3 Class Of Service Mask - COS 14 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=14.
		0:19		CBM: Bit vector of available L3 ways for COS 14 enforcement.
		63:20		Reserved
C9FH	3231	IA32_L3_QOS_MASK_15	Package	L3 Class Of Service Mask - COS 15 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=15.
		0:19		CBM: Bit vector of available L3 ways for COS 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 DisplayFamily_DisplayModel = 06_56H). The Intel® Xeon® processor D product family is based on the 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 DisplayFamily_DisplayModel 06_56H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ACH	428	MSR_TURBO_RATIO_LIMIT3	Package	Config Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		62:0	Package	Reserved
		63	Package	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).
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with DisplayFamily_DisplayModel 06_56H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC7 reports MC errors from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "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.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "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.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "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.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
See Table 2-20, Table 2-29, Table 2-34, and Table 2-36 for other MSR definitions applicable to processors with CPUID signature 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 Intel® Xeon® Processor E5 v4 and E7 v4 Families (CPUID DisplayFamily_DisplayModel = 06_4FH). The Intel® Xeon® processor E5 v4 family is based on the Broadwell micro-

architecture 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 DisplayFamily_DisplayModel 06_4FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ACH	428	MSR_TURBO_RATIO_LIMIT3	Package	Config Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0. RW if MSR_PLATFORM_INFO.[28] = 1.
		62:0	Package	Reserved
		63	Package	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).
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
294H	660	IA32_MC20_CTL2	Package	See Table 2-2.
295H	661	IA32_MC21_CTL2	Package	See Table 2-2.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC5 reports MC errors from the Intel QPI 0 module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_4FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC7 reports MC errors from the home agent HA 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC8 reports MC errors from the home agent HA 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with DisplayFamily_DisplayModel 06_4FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "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.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "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.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "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.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
450H	1104	IA32_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC20 reports MC errors from the Intel QPI 1 module.
451H	1105	IA32_MC20_STATUS	Package	
452H	1106	IA32_MC20_ADDR	Package	
453H	1107	IA32_MC20_MISC	Package	
454H	1108	IA32_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC21 reports MC errors from the Intel QPI 2 module.
455H	1109	IA32_MC21_STATUS	Package	
456H	1110	IA32_MC21_ADDR	Package	
457H	1111	IA32_MC21_MISC	Package	
C81H	3201	IA32_L3_QOS_CFG	Package	Cache Allocation Technology Configuration (R/W)
		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 CPUID signature 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 GENERATION, 7TH GENERATION, 8TH GENERATION, 9TH GENERATION, 10TH GENERATION, 11TH GENERATION, AND 12TH GENERATION INTEL® CORE™ PROCESSORS, INTEL® XEON® PROCESSOR SCALABLE FAMILY, 2ND AND 3RD GENERATION INTEL® XEON® PROCESSOR SCALABLE FAMILY, 8TH GENERATION INTEL® CORE™ I3 PROCESSORS, AND INTEL® XEON® E PROCESSORS

6th generation Intel® Core™ processors are based on the Skylake microarchitecture and have CPUID DisplayFamily_DisplayModel signatures of 06_4EH and 06_5EH.

The Intel® Xeon® Processor Scalable Family based on the Skylake microarchitecture, the 2nd generation Intel® Xeon® Processor Scalable Family based on the Cascade Lake product, and the 3rd generation Intel® Xeon® Processor Scalable Family based on the Cooper Lake product all have a CPUID DisplayFamily_DisplayModel signature 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 the Coffee Lake microarchitecture; these processors have CPUID DisplayFamily_DisplayModel signatures of 06_8EH and 06_9EH.

8th generation Intel® Core™ i3 processors are based on Cannon Lake microarchitecture and have a CPUID DisplayFamily_DisplayModel signature of 06_66H.

10th generation Intel® Core™ processors are based on Comet Lake microarchitecture (with CPUID DisplayFamily_DisplayModel signatures of 06_A5H, 06_A6H) and Ice Lake microarchitecture (with CPUID DisplayFamily_DisplayModel signatures of 06_7DH and 06_7EH).

11th generation Intel® Core™ processors are based on the Tiger Lake microarchitecture and have CPUID DisplayFamily_DisplayModel signatures of 06_8CH and 06_8DH.

The 3rd generation Intel® Xeon® Processor Scalable Family based on Ice Lake microarchitecture have CPUID DisplayFamily_DisplayModel signatures of 06_6AH and 06_6CH.

The 12th generation Intel® Core™ processors supporting the Alder Lake performance hybrid architecture have CPUID DisplayFamily_DisplayModel signatures of 06_97H, 06_9AH, and 06_BFH.

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 2-39[1]. For an MSR listed in Table 2-39 that also appears in the model-specific tables of prior generations, Table 2-39 supersede prior generation tables.

Tables 2-40 through 2-52 list additional supported MSR interfaces for 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.

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 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	Thread	MTRR Capability (RO, Architectural) See Table 2-2
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (RO) See Table 2-2.
		1		Thermal Status Log (R/WC0) See Table 2-2.
		2		PROTCHOT # or FORCEPR# Status (RO) See Table 2-2.
		3		PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.
		4		Critical Temperature Status (RO) See Table 2-2.
		5		Critical Temperature Status Log (R/WC0) See Table 2-2.
		6		Thermal threshold #1 Status (RO) See Table 2-2.
		7		Thermal threshold #1 Log (R/WC0) See Table 2-2.
		8		Thermal Threshold #2 Status (RO) See Table 2-2.
		9		Thermal Threshold #2 Log (R/WC0) See Table 2-2.
		10		Power Limitation Status (RO) See Table 2-2.
		11		Power Limitation Log (R/WC0) See Table 2-2.
		12		Current Limit Status (RO) See Table 2-2.
		13		Current Limit Log (R/WC0) See Table 2-2.
		14		Cross Domain Limit Status (RO) See Table 2-2.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		15		Cross Domain Limit Log (R/WC0) See Table 2-2.
		22:16		Digital Readout (RO) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (RO) See Table 2-2.
		31		Reading Valid (RO) See Table 2-2.
		63:32		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.
		15:8	Package	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.
		23:16	Package	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.
		31:24	Package	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.
		63:32		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Thread	Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register See http://biosbits.org.
		0		Reserved
		1	Package	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).
		18:2		Reserved

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
300H	768	MSR_SGXOWNEREPOCH0	Package	Lower 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.
301H	768	MSR_SGXOWNEREPOCH1	Package	Upper 64 Bit CR_SGXOWNEREPOCH (W) Writes do not update CR_SGXOWNEREPOCH if CPUID.(EAX=12H, ECX=0):EAX.SGX1 is 1 on any thread in the package.
		63:0		Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.
38EH	910	IA32_PERF_GLOBAL_STATUS		See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	Ovf_PMC0
		1	Thread	Ovf_PMC1
		2	Thread	Ovf_PMC2
		3	Thread	Ovf_PMC3
		4	Thread	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)
		5	Thread	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)
		6	Thread	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)
		7	Thread	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)
		31:8		Reserved
		32	Thread	Ovf_FixedCtr0

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		33	Thread	Ovf_FixedCtr1
		34	Thread	Ovf_FixedCtr2
		54:35		Reserved
		55	Thread	Trace_ToPA_PMI
		57:56		Reserved
		58	Thread	LBR_Frz
		59	Thread	CTR_Frz
		60	Thread	ASCI
		61	Thread	Ovf_Uncore
		62	Thread	Ovf_BufDSSAVE
		63	Thread	CondChgd
390H	912	IA32_PERF_GLOBAL_STATUS_RESET		See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	Set 1 to clear Ovf_PMC0.
		1	Thread	Set 1 to clear Ovf_PMC1.
		2	Thread	Set 1 to clear Ovf_PMC2.
		3	Thread	Set 1 to clear Ovf_PMC3.
		4	Thread	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).
		5	Thread	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).
		6	Thread	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).
		7	Thread	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to clear Ovf_FixedCtr0.
		33	Thread	Set 1 to clear Ovf_FixedCtr1.
		34	Thread	Set 1 to clear Ovf_FixedCtr2.
		54:35		Reserved
		55	Thread	Set 1 to clear Trace_ToPA_PMI.
		57:56		Reserved
		58	Thread	Set 1 to clear LBR_Frz.
		59	Thread	Set 1 to clear CTR_Frz.
		60	Thread	Set 1 to clear ASCI.
		61	Thread	Set 1 to clear Ovf_Uncore.
		62	Thread	Set 1 to clear Ovf_BufDSSAVE.
		63	Thread	Set 1 to clear CondChgd.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
391H	913	IA32_PERF_GLOBAL_STATUS_SET		See Table 2-2. See Section 19.2.4, "Architectural Performance Monitoring Version 4."
		0	Thread	Set 1 to cause Ovf_PMC0 = 1.
		1	Thread	Set 1 to cause Ovf_PMC1 = 1.
		2	Thread	Set 1 to cause Ovf_PMC2 = 1.
		3	Thread	Set 1 to cause Ovf_PMC3 = 1.
		4	Thread	Set 1 to cause Ovf_PMC4=1 (if CPUID.0AH:EAX[15:8] > 4).
		5	Thread	Set 1 to cause Ovf_PMC5=1 (if CPUID.0AH:EAX[15:8] > 5).
		6	Thread	Set 1 to cause Ovf_PMC6=1 (if CPUID.0AH:EAX[15:8] > 6).
		7	Thread	Set 1 to cause Ovf_PMC7=1 (if CPUID.0AH:EAX[15:8] > 7).
		31:8		Reserved
		32	Thread	Set 1 to cause Ovf_FixedCtr0 = 1.
		33	Thread	Set 1 to cause Ovf_FixedCtr1 = 1.
		34	Thread	Set 1 to cause Ovf_FixedCtr2 = 1.
		54:35		Reserved
		55	Thread	Set 1 to cause Trace_ToPA_PMI = 1.
		57:56		Reserved
		58	Thread	Set 1 to cause LBR_Frz = 1.
		59	Thread	Set 1 to cause CTR_Frz = 1.
		60	Thread	Set 1 to cause ASCI = 1.
		61	Thread	Set 1 to cause Ovf_Uncore.
		62	Thread	Set 1 to cause Ovf_BufDSSAVE.
		63		Reserved
392H	913	IA32_PERF_GLOBAL_INUSE	Thread	See Table 2-2.
3F7H	1015	MSR_PEBS_FRONTEND	Thread	FrontEnd Precise Event Condition Select (R/W)
		2:0		Event Code Select
		3		Reserved
		4		Event Code Select High
		7:5		Reserved
		19:8		IDQ_Bubble_Length Specifier
		22:20		IDQ_Bubble_Width Specifier
		63:23		Reserved

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
500H	1280	IA32_SGX_SVN_STATUS	Thread	Status and SVN Threshold of SGX Support for ACM (RO)
		0		Lock See Section 38.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		15:1		Reserved
		23:16		SGX_SVN_SINIT See Section 38.11.3, "Interactions with Authenticated Code Modules (ACMs)".
		63:24		Reserved
560H	1376	IA32_RTIT_OUTPUT_BASE	Thread	Trace Output Base Register (R/W) See Table 2-2.
561H	1377	IA32_RTIT_OUTPUT_MASK_PTRS	Thread	Trace Output Mask Pointers Register (R/W) See Table 2-2.
570H	1392	IA32_RTIT_CTL	Thread	Trace Control Register (R/W)
		0		TraceEn
		1		CYCEn
		2		OS
		3		User
		6:4		Reserved, must be zero.
		7		CR3 filter
		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
		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.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
571H	1393	IA32_RTIT_STATUS	Thread	Tracing Status Register (R/W)
		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.
572H	1394	IA32_RTIT_CR3_MATCH	Thread	Trace Filter CR3 Match Register (R/W)
		4:0		Reserved
		63:5		CR3[63:5] value to match
580H	1408	IA32_RTIT_ADDR0_A	Thread	Region 0 Start Address (R/W)
		63:0		See Table 2-2.
581H	1409	IA32_RTIT_ADDR0_B	Thread	Region 0 End Address (R/W)
		63:0		See Table 2-2.
582H	1410	IA32_RTIT_ADDR1_A	Thread	Region 1 Start Address (R/W)
		63:0		See Table 2-2.
583H	1411	IA32_RTIT_ADDR1_B	Thread	Region 1 End Address (R/W)
		63:0		See Table 2-2.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."
64DH	1613	MSR_PLATFORM_ENERGY_COUNTER	Platform*	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.
		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.Enery_Status_Unit.
		63:32		Reserved
64EH	1614	MSR_PPERF	Thread	Productive Performance Count (R/O)
		63:0		Hardware's view of workload scalability. See Section 14.4.5.1.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
64FH	1615	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		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.
		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.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
652H	1618	MSR_PKG_HDC_CONFIG	Package	HDC Configuration (R/W)
		2:0		PKG_Cx_Monitor Configures Package Cx state threshold for MSR_PKG_HDC_DEEP_RESIDENCY.
		63: 3		Reserved
653H	1619	MSR_CORE_HDC_RESIDENCY	Core	Core HDC Idle Residency (R/O)
		63:0		Core_Cx_Duty_Cycle_Cnt
655H	1621	MSR_PKG_HDC_SHALLOW_RESIDENCY	Package	Accumulate the cycles the package was in C2 state and at least one logical processor was in forced idle (R/O)
		63:0		Pkg_C2_Duty_Cycle_Cnt

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
656H	1622	MSR_PKG_HDC_DEEP_RESIDENCY	Package	Package Cx HDC Idle Residency (R/O)
		63:0		Pkg_Cx_Duty_Cycle_Cnt
658H	1624	MSR_WEIGHTED_CORE_C0	Package	Core-count Weighted C0 Residency (R/O)
		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.
659H	1625	MSR_ANY_CORE_C0	Package	Any Core C0 Residency (R/O)
		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.
65AH	1626	MSR_ANY_GFXE_C0	Package	Any Graphics Engine C0 Residency (R/O)
		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.
65BH	1627	MSR_CORE_GFXE_OVERLAP_C0	Package	Core and Graphics Engine Overlapped C0 Residency (R/O)
		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.
65CH	1628	MSR_PLATFORM_POWER_LIMIT	Platform*	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.
		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.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		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 (EAX=6):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, The unit is specified in MSR_RAPLPOWER_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.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
690H	1680	MSR_LASTBRANCH_16_FROM_IP	Thread	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 17.12.
691H	1681	MSR_LASTBRANCH_17_FROM_IP	Thread	Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
692H	1682	MSR_LASTBRANCH_18_FROM_IP	Thread	Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
693H	1683	MSR_LASTBRANCH_19_FROM_IP	Thread	Last Branch Record 19From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
694H	1684	MSR_LASTBRANCH_20_FROM_IP	Thread	Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
695H	1685	MSR_LASTBRANCH_21_FROM_IP	Thread	Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
696H	1686	MSR_LASTBRANCH_22_FROM_IP	Thread	Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
697H	1687	MSR_LASTBRANCH_23_FROM_IP	Thread	Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
698H	1688	MSR_LASTBRANCH_24_FROM_IP	Thread	Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
699H	1689	MSR_LASTBRANCH_25_FROM_IP	Thread	Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69AH	1690	MSR_LASTBRANCH_26_FROM_IP	Thread	Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69BH	1691	MSR_LASTBRANCH_27_FROM_IP	Thread	Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69CH	1692	MSR_LASTBRANCH_28_FROM_IP	Thread	Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69DH	1693	MSR_LASTBRANCH_29_FROM_IP	Thread	Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69EH	1694	MSR_LASTBRANCH_30_FROM_IP	Thread	Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.
69FH	1695	MSR_LASTBRANCH_31_FROM_IP	Thread	Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6B0H	1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)
		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 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
6B1H	1713	MSR_RING_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)
		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.
		15:12		Reserved

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		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.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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
6D0H	1744	MSR_LASTBRANCH_16_TO_IP	Thread	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 17.12.
6D1H	1745	MSR_LASTBRANCH_17_TO_IP	Thread	Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D2H	1746	MSR_LASTBRANCH_18_TO_IP	Thread	Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D3H	1747	MSR_LASTBRANCH_19_TO_IP	Thread	Last Branch Record 19To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D4H	1748	MSR_LASTBRANCH_20_TO_IP	Thread	Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D5H	1749	MSR_LASTBRANCH_21_TO_IP	Thread	Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D6H	1750	MSR_LASTBRANCH_22_TO_IP	Thread	Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D7H	1751	MSR_LASTBRANCH_23_TO_IP	Thread	Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D8H	1752	MSR_LASTBRANCH_24_TO_IP	Thread	Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6D9H	1753	MSR_LASTBRANCH_25_TO_IP	Thread	Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DAH	1754	MSR_LASTBRANCH_26_TO_IP	Thread	Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DBH	1755	MSR_LASTBRANCH_27_TO_IP	Thread	Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DCH	1756	MSR_LASTBRANCH_28_TO_IP	Thread	Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DDH	1757	MSR_LASTBRANCH_29_TO_IP	Thread	Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6DEH	1758	MSR_LASTBRANCH_30_TO_IP	Thread	Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
6DFH	1759	MSR_LASTBRANCH_31_TO_IP	Thread	Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.
770H	1904	IA32_PM_ENABLE	Package	See Section 14.4.2, "Enabling HWP".
771H	1905	IA32_HWP_CAPABILITIES	Thread	See Section 14.4.3, "HWP Performance Range and Dynamic Capabilities".
772H	1906	IA32_HWP_REQUEST_PKG	Package	See Section 14.4.4, "Managing HWP".
773H	1907	IA32_HWP_INTERRUPT	Thread	See Section 14.4.6, "HWP Notifications".
774H	1908	IA32_HWP_REQUEST	Thread	See Section 14.4.4, "Managing HWP".
		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
777H	1911	IA32_HWP_STATUS	Thread	See Section 14.4.5, "HWP Feedback".
D90H	3472	IA32_BNDCFGS	Thread	See Table 2-2.
DA0H	3488	IA32_XSS	Thread	See Table 2-2.
DB0H	3504	IA32_PKG_HDC_CTL	Package	See Section 14.5.2, "Package level Enabling HDC".
DB1H	3505	IA32_PM_CTL1	Thread	See Section 14.5.3, "Logical-Processor Level HDC Control".
DB2H	3506	IA32_THREAD_STALL	Thread	See Section 14.5.4.1, "IA32_THREAD_STALL".
DC0H	3520	MSR_LBR_INFO_0	Thread	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 17.9.1, "LBR Stack."
DC1H	3521	MSR_LBR_INFO_1	Thread	Last Branch Record 1 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC2H	3522	MSR_LBR_INFO_2	Thread	Last Branch Record 2 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC3H	3523	MSR_LBR_INFO_3	Thread	Last Branch Record 3 Additional Information (R/W) See description of MSR_LBR_INFO_0.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DC4H	3524	MSR_LBR_INFO_4	Thread	Last Branch Record 4 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC5H	3525	MSR_LBR_INFO_5	Thread	Last Branch Record 5 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC6H	3526	MSR_LBR_INFO_6	Thread	Last Branch Record 6 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC7H	3527	MSR_LBR_INFO_7	Thread	Last Branch Record 7 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC8H	3528	MSR_LBR_INFO_8	Thread	Last Branch Record 8 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DC9H	3529	MSR_LBR_INFO_9	Thread	Last Branch Record 9 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCAH	3530	MSR_LBR_INFO_10	Thread	Last Branch Record 10 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCBH	3531	MSR_LBR_INFO_11	Thread	Last Branch Record 11 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCCH	3532	MSR_LBR_INFO_12	Thread	Last Branch Record 12 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCDH	3533	MSR_LBR_INFO_13	Thread	Last Branch Record 13 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCEH	3534	MSR_LBR_INFO_14	Thread	Last Branch Record 14 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DCFH	3535	MSR_LBR_INFO_15	Thread	Last Branch Record 15 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD0H	3536	MSR_LBR_INFO_16	Thread	Last Branch Record 16 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD1H	3537	MSR_LBR_INFO_17	Thread	Last Branch Record 17 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD2H	3538	MSR_LBR_INFO_18	Thread	Last Branch Record 18 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD3H	3539	MSR_LBR_INFO_19	Thread	Last Branch Record 19 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD4H	3520	MSR_LBR_INFO_20	Thread	Last Branch Record 20 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD5H	3521	MSR_LBR_INFO_21	Thread	Last Branch Record 21 Additional Information (R/W) See description of MSR_LBR_INFO_0.

Table 2-39. Additional MSRs Supported by 6th Generation, 7th Generation, 8th Generation, 9th Generation, 10th Generation, 11th Generation, and 12th Generation Intel® Core™ Processors, Intel® Xeon® Processor Scalable Family, 2nd and 3rd Generation Intel® Xeon® Processor Scalable Family, 8th Generation Intel® Core™ i3 Processors, and Intel® Xeon® E Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
DD6H	3522	MSR_LBR_INFO_22	Thread	Last Branch Record 22 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD7H	3523	MSR_LBR_INFO_23	Thread	Last Branch Record 23 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD8H	3524	MSR_LBR_INFO_24	Thread	Last Branch Record 24 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DD9H	3525	MSR_LBR_INFO_25	Thread	Last Branch Record 25 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDAH	3526	MSR_LBR_INFO_26	Thread	Last Branch Record 26 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDBH	3527	MSR_LBR_INFO_27	Thread	Last Branch Record 27 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDCH	3528	MSR_LBR_INFO_28	Thread	Last Branch Record 28 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDDH	3529	MSR_LBR_INFO_29	Thread	Last Branch Record 29 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDEH	3530	MSR_LBR_INFO_30	Thread	Last Branch Record 30 Additional Information (R/W) See description of MSR_LBR_INFO_0.
DDFH	3531	MSR_LBR_INFO_31	Thread	Last Branch Record 31 Additional Information (R/W) See description of MSR_LBR_INFO_0.

Table 2-40 lists the MSRs of uncore PMU for Intel processors with CPUID DisplayFamily_DisplayModel signatures of 06_4EH, 06_5EH, 06_8EH, 06_9EH, and 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		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		43:0		Current count.

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		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		63:44		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSEL0	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb Unit, Counter 1 Event Select MSR
700H	1792	MSR_UNC_CBO_0_PERFEVTSEL0	Package	Uncore C-Box 0, Counter 0 Event Select MSR
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
706H	1798	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, Performance Counter 0
707H	1799	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
710H	1808	MSR_UNC_CBO_1_PERFEVTSEL0	Package	Uncore C-Box 1, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
716H	1814	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, Performance Counter 0
717H	1815	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
720H	1824	MSR_UNC_CBO_2_PERFEVTSEL0	Package	Uncore C-Box 2, Counter 0 Event Select MSR
721H	1825	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
726H	1830	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, Performance Counter 0
727H	1831	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
730H	1840	MSR_UNC_CBO_3_PERFEVTSEL0	Package	Uncore C-Box 3, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
736H	1846	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, Performance Counter 0
737H	1847	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
E01H	3585	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		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

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		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
E02H	3586	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		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 DisplayFamily_DisplayModel signatures of 06_8EH and 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 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
80H	128	MSR_TRACE_HUB_STH_ACPIBAR_BASE	Package	NPK Address Used by AET Messages (R/W)
		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.
		17:1		Reserved
		63:18		ACPIBAR_BASE_ADDRESS AET target address in NPK MMIO space.
1F4H	500	MSR_PRMRR_PHYS_BASE	Core	Processor Reserved Memory Range Register - Physical Base Control Register (R/W)
		2:0		MemType PRMRR BASE MemType.
		11:3		Reserved
		45:12		Base PRMRR Base Address.
		63:46		Reserved
1F5H	501	MSR_PRMRR_PHYS_MASK	Core	Processor Reserved Memory Range Register - Physical Mask Control Register (R/W)
		9:0		Reserved
		10		Lock Lock bit for the PRMRR.

Table 2-41. Additional MSRs Supported by 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		11		VLD Enable bit for the PRMRR.
		45:12		Mask PRMRR MASK bits.
		63:46		Reserved
1FBH	507	MSR_PRMRR_VALID_CONFIG	Core	Valid PRMRR Configurations (R/W)
		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
2F4H	756	MSR_UNCORE_PRMRR_PHYS_BASE[1]	Package	(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
		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
2F5H	757	MSR_UNCORE_PRMRR_PHYS_MASK[1]	Package	(R/W) This register controls the size of the PRMRR range by indicating which address bits must match the PRMRR base register value.
		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

Table 2-41. Additional MSRs Supported by 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
620H	1568	MSR_RING_RATIO_LIMIT	Package	Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.
		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 DisplayFamily_DisplayModel signature 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 supersede prior generation tables.

Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		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. Available only if CPUID.(EAX=07H, ECX=0H): ECX[30] = 1.
		18		SGX Global Functions Enable (R/WL)
		63:21		Reserved
350H	848	MSR_BR_DETECT_CTRL		Branch Monitoring Global Control (R/W)

Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
		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
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.

Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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).
		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

Table 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
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
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Package C3 Residency Counter (R/O)
		63:0		Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
620H	1568	MSR_RING_RATIO_LIMIT	Package	Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.
		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 2-42. Additional MSRs Supported by 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture (Contd.)

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
660H	1632	MSR_CORE_C1_RESIDENCY	Core	Core C1 Residency Counter (R/O)
		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.
662H	1634	MSR_CORE_C3_RESIDENCY	Core	Core C3 Residency Counter (R/O)
		63:0		Will always return 0.

Table 2-43 lists the MSRs of uncore PMU for Intel processors with CPUID signature 06_66H.

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		47:0		Current count.
		63:48		Reserved
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Report the number of C-Box units with performance counters, including processor cores and processor graphics.
		63:4		Reserved
3B0H	946	MSR_UNC_ARB_PERFCTR0	Package	Uncore Arb Unit, Performance Counter 0
3B1H	947	MSR_UNC_ARB_PERFCTR1	Package	Uncore Arb Unit, Performance Counter 1
3B2H	944	MSR_UNC_ARB_PERFEVTSEL0	Package	Uncore Arb Unit, Counter 0 Event Select MSR
3B3H	945	MSR_UNC_ARB_PERFEVTSEL1	Package	Uncore Arb unit, Counter 1 Event Select MSR
700H	1792	MSR_UNC_CBO_0_PERFEVTSEL0	Package	Uncore C-Box 0, Counter 0 Event Select MSR

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
701H	1793	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR
702H	1794	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, Performance Counter 0
703H	1795	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
708H	1800	MSR_UNC_CBO_1_PERFEVTSEL0	Package	Uncore C-Box 1, Counter 0 Event Select MSR
709H	1801	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
70AH	1802	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, Performance Counter 0
70BH	1803	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
710H	1808	MSR_UNC_CBO_2_PERFEVTSEL0	Package	Uncore C-Box 2, Counter 0 Event Select MSR
711H	1809	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
712H	1810	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, Performance Counter 0
713H	1811	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
718H	1816	MSR_UNC_CBO_3_PERFEVTSEL0	Package	Uncore C-Box 3, Counter 0 Event Select MSR
719H	1817	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
71AH	1818	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, Performance Counter 0
71BH	1819	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
720H	1824	MSR_UNC_CBO_4_PERFEVTSEL0	Package	Uncore C-Box 4, Counter 0 Event Select MSR
721H	1825	MSR_UNC_CBO_4_PERFEVTSEL1	Package	Uncore C-Box 4, Counter 1 Event Select MSR
722H	1826	MSR_UNC_CBO_4_PERFCTR0	Package	Uncore C-Box 4, Performance Counter 0
723H	1827	MSR_UNC_CBO_4_PERFCTR1	Package	Uncore C-Box 4, Performance Counter 1
728H	1832	MSR_UNC_CBO_5_PERFEVTSEL0	Package	Uncore C-Box 5, Counter 0 Event Select MSR
729H	1833	MSR_UNC_CBO_5_PERFEVTSEL1	Package	Uncore C-Box 5, Counter 1 Event Select MSR
72AH	1834	MSR_UNC_CBO_5_PERFCTR0	Package	Uncore C-Box 5, Performance Counter 0
72BH	1835	MSR_UNC_CBO_5_PERFCTR1	Package	Uncore C-Box 5, Performance Counter 1
730H	1840	MSR_UNC_CBO_6_PERFEVTSEL0	Package	Uncore C-Box 6, Counter 0 Event Select MSR
731H	1841	MSR_UNC_CBO_6_PERFEVTSEL1	Package	Uncore C-Box 6, Counter 1 Event Select MSR
732H	1842	MSR_UNC_CBO_6_PERFCTR0	Package	Uncore C-Box 6, Performance Counter 0
733H	1843	MSR_UNC_CBO_6_PERFCTR1	Package	Uncore C-Box 6, Performance Counter 1
738H	1848	MSR_UNC_CBO_7_PERFEVTSEL0	Package	Uncore C-Box 7, Counter 0 Event Select MSR
739H	1849	MSR_UNC_CBO_7_PERFEVTSEL1	Package	Uncore C-Box 7, Counter 1 Event Select MSR
73AH	1850	MSR_UNC_CBO_7_PERFCTR0	Package	Uncore C-Box 7, Performance Counter 0
73BH	1851	MSR_UNC_CBO_7_PERFCTR1	Package	Uncore C-Box 7, Performance Counter 1
E01H	3585	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		0		Slice 0 select.
		1		Slice 1 select.
		2		Slice 2 select.
		3		Slice 3 select.
		4		Slice 4select.

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		18:5		Reserved
		29		Enable all uncore counters.
		30		Enable wake on PMI.
		31		Enable Freezing counter when overflow.
		63:32		Reserved
E02H	3586	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		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 DisplayFamily_DisplayModel signature values of 06_7DH and 06_7EH. For an MSR listed in Table 2-44 that also appears in the model-specific tables of prior generations, Table 2-44 supersede prior generation tables.

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
33H	51	MSR_MEMORY_CTRL	Core	Memory Control Register
		28:0		Reserved.
		29		Enable #AC(0) exception for split locked accesses: Cause #AC(0) exception for split locked access at all CPL irrespective of CR0.AM or EFLAGS.AC. If bits 29 and 31 are both set, bit 29 takes precedence.
		30		Reserved.
		31		Reserved.
48H	72	IA32_SPEC_CTRL	Core	See Table 2-2.
49H	73	IA32_PREDICT_CMD	Thread	See Table 2-2.
8CH	140	IA32_SGXLEPUBKEYHASH0	Thread	See Table 2-2.
8DH	141	IA32_SGXLEPUBKEYHASH1	Thread	See Table 2-2.
8EH	142	IA32_SGXLEPUBKEYHASH2	Thread	See Table 2-2.
8FH	143	IA32_SGXLEPUBKEYHASH3	Thread	See Table 2-2.
A0H	160	MSR_BIOS_MCU_ERRORCODE	Package	BIOS MCU ERRORCODE (R/O) This MSR indicates if WRMSR 0x79 failed to configure PRM memory and gives a hint to debug BIOS.
		15:0	Package	Error Codes (R/O)

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		30:16		Reserved.
		31	Thread	MCU Partial Success (R/O) When set to 1, WRMSR 0x79 skipped part of the functionality during BIOS.
A5H	165	MSR_FIT_BIOS_ERROR	Thread	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.
		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.
10BH	267	IA32_FLUSH_CMD	Thread	See Table 2-2.
151H	337	MSR_BIOS_DONE	Thread	BIOS Done (R/WO)
		0	Thread	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.
		1	Package	Package BIOS Done Indication (R/O) When set to 1, all threads in the package have bit 0 of this MSR set.
		31:2		Reserved.
1F1H	497	MSR_CRASHLOG_CONTROL	Thread	Write Data to a Crash Log Configuration
		0		CDDIS: CrashDump_Disable If set, indicates that Crash Dump is disabled.
		63:1		Reserved.
2A0H	672	MSR_PRMRR_BASE_0	Core	Processor Reserved Memory Range Register - Physical Base Control Register (R/W)
		2:0		MEMTYPE: PRMRR BASE Memory Type.
		3		CONFIGURED: PRMRR BASE Configured.
		11:4		Reserved.
		51:12		BASE: PRMRR Base Address.
		63:52		Reserved.

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
30CH	780	IA32_FIXED_CTR3	Thread	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.
329H	809	MSR_PERF_METRICS	Thread	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).
		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.
		63:25		Reserved.
3F2H	1010	MSR_PEBS_DATA_CFG	Thread	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.
		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.
		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.

Table 2-44. MSRs Supported by 10th Generation Intel® Core™ Processors Based on Ice Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
541H	1345	MSR_CORE_UARCH_CTL	Core	Core Microarchitecture Control MSR (R/W)
		0		L1 Scrubbing Enable When set to 1, enable L1 scrubbing.
		31:1		Reserved.
657H	1623	MSR_FAST_UNCORE_MSRS_CTL	Thread	Fast WRMSR/RDMSR Control MSR (R/W)
		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[6].EAX[18] to be set.
		31:4		Reserved.
65EH	1630	MSR_FAST_UNCORE_MSRS_STATUS	Thread	Indication of Uncore MSRs, Post Write Activates
		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.
65FH	1631	MSR_FAST_UNCORE_MSRS_CAPABILITY	Thread	Fast WRMSR/RDMSR Enumeration MSR (RO)
		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.
772H	1906	IA32_HWP_REQUEST_PKG	Package	See Table 2-2.
775H	1909	IA32_PECI_HWP_REQUEST_INFO	Thread	See Table 2-2.
777H	1911	IA32_HWP_STATUS	Thread	See Table 2-2.

2.17.4 MSRs Introduced in 11th Generation Intel® Core™ Processors based on Tiger Lake Microarchitecture

Table 2-45 lists additional MSRs for 11th generation Intel Core processors with CPUID DisplayFamily_DisplayModel signatures of 06_8CH and 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 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
A0H	160	MSR_BIOS_MCU_ERRORCODE	Package	BIOS MCU ERRORCODE (R/O)
		15:0		Error Codes
		31:16		Reserved
A7H	167	MSR_BIOS_DEBUG	Thread	BIOS DEBUG (R/O) This MSR indicates if WRMSR 79H failed to configure PRM memory and gives a hint to debug BIOS.
		30:0		Reserved
		31		MCU Partial Success When set to 1, WRMSR 79H skipped part of the functionality during BIOS.
		63:32		Reserved
CFH	207	IA32_CORE_CAPABILITIES	Package	IA32 Core Capabilities Register (R/O) If CPUID.(EAX=07H, ECX=0):EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.
		1:0		Reserved
		2		FUSA_SUPPORTED
		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 set to 1, the ability to set MEMORY_CONTROL (MSR 33H) bit 29 enables an #AC to be created when a split lock is detected.
		31:6		Reserved

Table 2-45. Additional MSRs Supported by 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
492H	1170	IA32_VMX_PROCBASED_CTLS3	Core	IA32_VMX_PROCBASED_CTLS3 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.
		0		LOADIWKEY This control determines whether executions of LOADIWKEY cause VM exits.
		63:1		Reserved
601H	1537	MSR_VR_CURRENT_CONFIG	Package	Power Limit 4 (PL4) Package-level maximum power limit (in Watts). It is a proactive, instantaneous limit.
		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
6A0H	1696	IA32_U_CET		Configure User Mode CET (R/W) See Table 2-2.
6A2H	1698	IA32_S_CET		Configure Supervisor Mode CET (R/W) See Table 2-2.
6A4H	1700	IA32_PL0_SSP		Linear address to be loaded into SSP on transition to privilege level 0. (R/W) See Table 2-2.
6A5H	1701	IA32_PL1_SSP		Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.
6A6H	1702	IA32_PL2_SSP		Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.

Table 2-45. Additional MSRs Supported by 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
6A7H	1703	IA32_PL3_SSP		Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.
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.
981H	2433	IA32_TME_CAPABILITY		See Table 2-2.
982H	2434	IA32_TME_ACTIVATE		See Table 2-2.
983H	2435	IA32_TME_EXCLUDE_MASK		See Table 2-2.
984H	2436	IA32_TME_EXCLUDE_BASE		See Table 2-2.
990H	2448	IA32_COPY_STATUS[1]	Thread	See Table 2-2.
991H	2449	IA32_IWKEYBACKUP_STATUS[1]	Platform	See Table 2-2.
C82H	3202	IA32_L2_QOS_CFG	Core	IA32_CR_L2_QOS_CFG This MSR provides software an enumeration of the parameters that L2 QoS (Intel RDT) support in any particular implementation.
		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 17, "Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features" for further details on CDP.
		31:1		Reserved
D10H - D17H	3220 - 3351	IA32_L2_QOS_MASK_[0-7]	Package	IA32_CR_L2_QOS_MASK_[0-7] Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 17, "Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features".
		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.4.2: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
D91H	3473	IA32_COPY_LOCAL_TO_PLATFORM[1]	Thread	See Table 2-2.
D92H	3474	IA32_COPY_PLATFORM_TO_LOCAL[1]	Thread	See Table 2-2.

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 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Table 2-47 lists additional MSRs for 12th generation Intel Core processors with CPUID DisplayFamily_DisplayModel signatures of 06_97H, 06_9AH, and 06_BFH. Table 2-48 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 2-44[1] 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 2-48, Table 2-47 and Table 2-48 supersede prior generation tables.

Table 2-46. Additional MSRs Supported by 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
33H	51	MSR_MEMORY_CTRL	Core	Memory Control Register
		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 load lock will trigger a #GP fault.
		29		Enable #AC(0) exception for split locked accesses: Cause #AC(0) exception for split locked access at all CPL irrespective of CR0.AM or EFLAGS.AC. If bits 29 and 31 are both set, bit 29 takes precedence.
		30		Reserved.
		31		Reserved.
BCH	188	IA32_MISC_PACKAGE_CTLS	Package	Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.
C7H	199	IA32_PMC6	Core	General Performance Counter 6 (R/W) See Table 2-2.
C8H	200	IA32_PMC7	Core	General Performance Counter 7 (R/W) See Table 2-2.

1. MSRs at the following addresses are not supported in the 12th generation Intel Core processor E-core: 30CH, 329H, 541H and 657H. The MSR at address 657H is not supported in the 12th generation Intel Core processor P-core.

Table 2-46. Additional MSRs Supported by 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CFH	207	IA32_CORE_CAPABILITIES	Package	IA32 Core Capabilities Register (R/O) If CPUID.(EAX=07H, ECX=0):EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.
		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_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		UC_LOCK_DISABLE_SUPPORTED When set to 1, processor supports UC load lock disable feature.
		5		SPLIT_LOCK_DISABLE_SUPPORTED When set to 1, the ability to set MEMORY_CONTROL (MSR 33H) bit 29 enables an #AC to be created when a split lock is detected.
		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.
		7		UC_STORE_THROTTLING_SUPPORTED When set 1, UC Store throttle capability exist through MSR_MEMORY_CTRL (33H) bit 27.
		31:8		Reserved
E1H	225	IA32_UMWAIT_CONTROL		UMWAIT Control (R/W) See Table 2-2.
10AH	266	IA32_ARCH_CAPABILITIES		Enumeration of Architectural Features (R/O) See Table 2-2.
18CH	396	IA32_PERFEVTSEL6	Core	See Table 2-20.
18DH	397	IA32_PERFEVTSEL7	Core	See Table 2-20.

Table 2-46. Additional MSRs Supported by 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_PRIMARY_TURBO_RATIO_LIMIT	Package	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.
		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.
493H	1171	IA32_VMX_EXIT_CTLS2		See Table 2-2.
4C7H	1223	IA32_A_PMC6		Full Width Writable IA32_PMC6 Alias (R/W) See Table 2-2.
4C8H	1224	IA32_A_PMC7		Full Width Writable IA32_PMC7 Alias (R/W) See Table 2-2.
650H	1616	MSR_SECONDARY_TURBO_RATIO_LIMIT	Package	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.
		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 2-46. Additional MSRs Supported by 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
664H	1636	MSR_MC6_RESIDENCY_COUNTER	Module	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.
		63:0		Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.
6E1H	1761	IA32_PKRS		Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.
776H	1910	IA32_HWP_CTL		See Table 2-2.
981H	2433	IA32_TME_CAPABILITY		Memory Encryption Capability MSR See Table 2-2.
1200H - 121FH	4608 - 4639	IA32_LBR_x_INFO		Last Branch Record Entry X Info Register (R/W) See Table 2-2.
14CEH	5326	IA32_LBR_CTL		Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.
14CFH	5327	IA32_LBR_DEPTH		Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.
1500H - 151FH	5376 - 5407	IA32_LBR_x_FROM_IP		Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.
1600H - 161FH	5632 - 5663	IA32_LBR_x_TO_IP		Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.
17D2H	6098	IA32_THREAD_FEEDBACK_CHAR		Thread Feedback Characteristics (R/O) See Table 2-2.
17D4H	6100	IA32_HW_FEEDBACK_THREAD_CONFIG		Hardware Feedback Thread Configuration (R/W) See Table 2-2.

Table 2-46. Additional MSRs Supported by 12th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
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 generation Intel Core processor P-core. These MSRs are not supported on the processor E-core.

Table 2-47. MSRs Supported by 12th Generation Intel® Core™ Processor P-core

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
3F7H	1015	MSR_PEBS_FRONTEND	Thread	FrontEnd Precise Event Condition Select (R/W) See Table 2-39.
540H	1344	MSR_THREAD_UARCH_CTL	Thread	Thread Microarchitectural Control (R/W)
		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
541H	1345	MSR_CORE_UARCH_CTL	Core	Core Microarchitecture Control MSR (R/W) See Table 2-44.
D10H - D17H	3220 - 3351	IA32_L2_QOS_MASK_[0-7]	Core	IA32_CR_L2_QOS_MASK_[0-7] If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 17, "Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features".
		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.4.2: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 generation Intel Core processor E-core. These MSRs are not supported on the processor P-core.

Table 2-48. MSRs Supported by 12th Generation Intel® Core™ Processor E-core

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
D10H - D1FH	3220 - 3359	IA32_L2_QOS_MASK_[0-15]	Module	IA32_CR_L2_QOS_MASK_[0-15] If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 17, "Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features".
		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.4.2: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
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
14C1H - 14C6H	5313 - 5318	MSR_RELOAD_PMCx	Core	Reload value for IA32_PMCx (R/W)
		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

Table 2-49 lists the MSRs of uncore PMU for Intel processors with CPUID DisplayFamily_DisplayModel signatures of 06_97H, 06_9AH, and 06_BFH.

Table 2-49. Uncore PMU MSRs Supported by 12th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
396H	918	MSR_UNC_CBO_CONFIG	Package	Uncore C-Box Configuration Information (R/O)
		3:0		Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).
		63:4		Reserved
2000H	8192	MSR_UNC_CBO_0_PERFEVTSEL0	Package	Uncore C-Box 0, Counter 0 Event Select MSR
2001H	8193	MSR_UNC_CBO_0_PERFEVTSEL1	Package	Uncore C-Box 0, Counter 1 Event Select MSR

Table 2-49. Uncore PMU MSRs Supported by 12th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
2002H	8194	MSR_UNC_CBO_0_PERFCTR0	Package	Uncore C-Box 0, Performance Counter 0
2003H	8195	MSR_UNC_CBO_0_PERFCTR1	Package	Uncore C-Box 0, Performance Counter 1
2008H	8200	MSR_UNC_CBO_1_PERFEVTSEL0	Package	Uncore C-Box 1, Counter 0 Event Select MSR
2009H	8201	MSR_UNC_CBO_1_PERFEVTSEL1	Package	Uncore C-Box 1, Counter 1 Event Select MSR
200AH	8202	MSR_UNC_CBO_1_PERFCTR0	Package	Uncore C-Box 1, Performance Counter 0
200BH	8203	MSR_UNC_CBO_1_PERFCTR1	Package	Uncore C-Box 1, Performance Counter 1
2010H	8208	MSR_UNC_CBO_2_PERFEVTSEL0	Package	Uncore C-Box 2, Counter 0 Event Select MSR
2011H	8209	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, Counter 1 Event Select MSR
2012H	8210	MSR_UNC_CBO_2_PERFCTR0	Package	Uncore C-Box 2, Performance Counter 0
2013H	8211	MSR_UNC_CBO_2_PERFCTR1	Package	Uncore C-Box 2, Performance Counter 1
2018H	8216	MSR_UNC_CBO_3_PERFEVTSEL0	Package	Uncore C-Box 3, Counter 0 Event Select MSR
2019H	8217	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, Counter 1 Event Select MSR
201AH	8218	MSR_UNC_CBO_3_PERFCTR0	Package	Uncore C-Box 3, Performance Counter 0
201BH	8219	MSR_UNC_CBO_3_PERFCTR1	Package	Uncore C-Box 3, Performance Counter 1
2020H	8224	MSR_UNC_CBO_4_PERFEVTSEL0	Package	Uncore C-Box 4, Counter 0 Event Select MSR
2021H	8225	MSR_UNC_CBO_4_PERFEVTSEL1	Package	Uncore C-Box 4, Counter 1 Event Select MSR
2022H	8226	MSR_UNC_CBO_4_PERFCTR0	Package	Uncore C-Box 4, Performance Counter 0
2023H	8227	MSR_UNC_CBO_4_PERFCTR1	Package	Uncore C-Box 4, Performance Counter 1
2028H	8232	MSR_UNC_CBO_5_PERFEVTSEL0	Package	Uncore C-Box 5, Counter 0 Event Select MSR
2029H	8233	MSR_UNC_CBO_5_PERFEVTSEL1	Package	Uncore C-Box 5, Counter 1 Event Select MSR
202AH	8234	MSR_UNC_CBO_5_PERFCTR0	Package	Uncore C-Box 5, Performance Counter 0
202BH	8235	MSR_UNC_CBO_5_PERFCTR1	Package	Uncore C-Box 5, Performance Counter 1
2030H	8240	MSR_UNC_CBO_6_PERFEVTSEL0	Package	Uncore C-Box 6, Counter 0 Event Select MSR
2031H	8241	MSR_UNC_CBO_6_PERFEVTSEL1	Package	Uncore C-Box 6, Counter 1 Event Select MSR
2032H	8242	MSR_UNC_CBO_6_PERFCTR0	Package	Uncore C-Box 6, Performance Counter 0
2033H	8243	MSR_UNC_CBO_6_PERFCTR1	Package	Uncore C-Box 6, Performance Counter 1
2038H	8248	MSR_UNC_CBO_7_PERFEVTSEL0	Package	Uncore C-Box 7, Counter 0 Event Select MSR
2039H	8249	MSR_UNC_CBO_7_PERFEVTSEL1	Package	Uncore C-Box 7, Counter 1 Event Select MSR
203AH	8250	MSR_UNC_CBO_7_PERFCTR0	Package	Uncore C-Box 7, Performance Counter 0
203BH	8251	MSR_UNC_CBO_7_PERFCTR1	Package	Uncore C-Box 7, Performance Counter 1
2040H	8256	MSR_UNC_CBO_8_PERFEVTSEL0	Package	Uncore C-Box 8, Counter 0 Event Select MSR
2041H	8257	MSR_UNC_CBO_8_PERFEVTSEL1	Package	Uncore C-Box 8, Counter 1 Event Select MSR
2042H	8258	MSR_UNC_CBO_8_PERFCTR0	Package	Uncore C-Box 8, Performance Counter 0
2043H	8259	MSR_UNC_CBO_8_PERFCTR1	Package	Uncore C-Box 8, Performance Counter 1
2048H	8264	MSR_UNC_CBO_9_PERFEVTSEL0	Package	Uncore C-Box 9, Counter 0 Event Select MSR
2049H	8265	MSR_UNC_CBO_9_PERFEVTSEL1	Package	Uncore C-Box 9, Counter 1 Event Select MSR
204AH	8266	MSR_UNC_CBO_9_PERFCTR0	Package	Uncore C-Box 9, Performance Counter 0

Table 2-49. Uncore PMU MSRs Supported by 12th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
204BH	8267	MSR_UNC_CBO_9_PERFCTR1	Package	Uncore C-Box 9, Performance Counter 1
2FD0H	12240	MSR_UNC_ARB_0_PERFEVTSEL0	Package	Uncore Arb Unit 0, Counter 0 Event Select MSR
2FD1H	12241	MSR_UNC_ARB_0_PERFEVTSEL1	Package	Uncore Arb Unit 0, Counter 1 Event Select MSR
2FD2H	12242	MSR_UNC_ARB_0_PERFCTR0	Package	Uncore Arb Unit 0, Performance Counter 0
2FD3H	12243	MSR_UNC_ARB_0_PERFCTR1	Package	Uncore Arb Unit 0, Performance Counter 1
2FD4H	12244	MSR_UNC_ARB_0_PERF_STATUS	Package	Uncore Arb Unit 0, Performance Status
2FD5H	12245	MSR_UNC_ARB_0_PERF_CTRL	Package	Uncore Arb Unit 0, Performance Control
2FD8H	12248	MSR_UNC_ARB_1_PERFEVTSEL0	Package	Uncore Arb Unit 1, Counter 0 Event Select MSR
2FD9H	12249	MSR_UNC_ARB_1_PERFEVTSEL1	Package	Uncore Arb Unit 1, Counter 1 Event Select MSR
2FDAH	12250	MSR_UNC_ARB_1_PERFCTR0	Package	Uncore Arb Unit 1, Performance Counter 0
2FDBH	12251	MSR_UNC_ARB_1_PERFCTR1	Package	Uncore Arb Unit 1, Performance Counter 1
2FDCH	12252	MSR_UNC_ARB_1_PERF_STATUS	Package	Uncore Arb Unit 1, Performance Status
2FDDH	12253	MSR_UNC_ARB_1_PERF_CTRL	Package	Uncore Arb Unit 1, Performance Control
2FDEH	12254	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore Fixed Counter Control (R/W)
		19:0		Reserved
		20		Enable overflow propagation.
		21		Reserved
		22		Enable counting.
		63:23		Reserved
2FDFH	12255	MSR_UNC_PERF_FIXED_CTR	Package	Uncore Fixed Counter
		43:0		Current count.
		63:44		Reserved
2FF0H	12272	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU Global Control
		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
2FF2H	12274	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU Main Status
		0		Fixed counter overflowed.
		1		An ARB counter overflowed.
		2		Reserved

Table 2-49. Uncore PMU MSRs Supported by 12th Generation Intel® Core™ Processors

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		3		A CBox counter overflowed (on any slice).
		63:4		Reserved

2.17.6 MSRs Introduced in Intel® Xeon® Processor Scalable Family

Intel® Xeon® Processor Scalable Family (CPUID DisplayFamily_DisplayModel = 06_55H) support the MSRs listed in Table 2-50.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64 Processor (R/W) See Table 2-2.
		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)
		63:21		Reserved
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) See Table 2-26.
		22:16		Reserved.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		23	Package	PPIN_CAP (R/O) See Table 2-26.
		27:24		Reserved
		28	Package	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.
		29	Package	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.
		30	Package	Programmable TJ OFFSET (R/O) See Table 2-26.
		39:31		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) See Table 2-26.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	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.
		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 MWAT(C1) to MWAIT(C6).
		24:17		Reserved
		25		C3 State Auto Demotion Enable (R/W)
		26		C1 State Auto Demotion Enable (R/W)

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		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
179H	377	IA32_MCG_CAP	Thread	Global Machine Check Capability (R/O)
		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
17DH	381	MSR_SMM_MCA_CAP	THREAD	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		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
19CH	412	IA32_THERM_STATUS	Core	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (RO) See Table 2-2.
		1		Thermal Status Log (R/WC0) See Table 2-2.
		2		PROTCHOT # or FORCEPR# Status (RO) See Table 2-2.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		3		PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.
		4		Critical Temperature Status (RO) See Table 2-2.
		5		Critical Temperature Status Log (R/WC0) See Table 2-2.
		6		Thermal Threshold #1 Status (RO) See Table 2-2.
		7		Thermal Threshold #1 Log (R/WC0) See Table 2-2.
		8		Thermal Threshold #2 Status (RO) See Table 2-2.
		9		Thermal Threshold #2 Log (R/WC0) See Table 2-2.
		10		Power Limitation Status (RO) See Table 2-2.
		11		Power Limitation Log (R/WC0) See Table 2-2.
		12		Current Limit Status (RO) See Table 2-2.
		13		Current Limit Log (R/WC0) See Table 2-2.
		14		Cross Domain Limit Status (RO) See Table 2-2.
		15		Cross Domain Limit Log (R/WC0) See Table 2-2.
		22:16		Digital Readout (RO) See Table 2-2.
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (RO) See Table 2-2.
		31		Reading Valid (RO) See Table 2-2.
		63:32		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (RO) See Table 2-26.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		27:24		TCC Activation Offset (R/W) See Table 2-26.
		63:28		Reserved
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	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: ▪ 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.
		55:48		RATIO_6 Defines ratio limits.
		63:56		RATIO_7 Defines ratio limits.
1AEH	430	MSR_TURBO_RATIO_LIMIT_CORES	Package	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.
		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.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		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.
280H	640	IA32_MC0_CTL2	Core	See Table 2-2.
281H	641	IA32_MC1_CTL2	Core	See Table 2-2.
282H	642	IA32_MC2_CTL2	Core	See Table 2-2.
283H	643	IA32_MC3_CTL2	Core	See Table 2-2.
284H	644	IA32_MC4_CTL2	Package	See Table 2-2.
285H	645	IA32_MC5_CTL2	Package	See Table 2-2.
286H	646	IA32_MC6_CTL2	Package	See Table 2-2.
287H	647	IA32_MC7_CTL2	Package	See Table 2-2.
288H	648	IA32_MC8_CTL2	Package	See Table 2-2.
289H	649	IA32_MC9_CTL2	Package	See Table 2-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 2-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 2-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 2-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 2-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 2-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 2-2.
290H	656	IA32_MC16_CTL2	Package	See Table 2-2.
291H	657	IA32_MC17_CTL2	Package	See Table 2-2.
292H	658	IA32_MC18_CTL2	Package	See Table 2-2.
293H	659	IA32_MC19_CTL2	Package	See Table 2-2.
400H	1024	IA32_MC0_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC0 reports MC errors from the IFU module.
401H	1025	IA32_MC0_STATUS	Core	
402H	1026	IA32_MC0_ADDR	Core	
403H	1027	IA32_MC0_MISC	Core	

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
404H	1028	IA32_MC1_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC1 reports MC errors from the DCU module.
405H	1029	IA32_MC1_STATUS	Core	
406H	1030	IA32_MC1_ADDR	Core	
407H	1031	IA32_MC1_MISC	Core	
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC2 reports MC errors from the DTLB module.
409H	1033	IA32_MC2_STATUS	Core	
40AH	1034	IA32_MC2_ADDR	Core	
40BH	1035	IA32_MC2_MISC	Core	
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC3 reports MC errors from the MLC module.
40DH	1037	IA32_MC3_STATUS	Core	
40EH	1038	IA32_MC3_ADDR	Core	
40FH	1039	IA32_MC3_MISC	Core	
410H	1040	IA32_MC4_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC4 reports MC errors from the PCU module.
411H	1041	IA32_MC4_STATUS	Package	
412H	1042	IA32_MC4_ADDR	Package	
413H	1043	IA32_MC4_MISC	Package	
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC5 reports MC errors from a link interconnect module.
415H	1045	IA32_MC5_STATUS	Package	
416H	1046	IA32_MC5_ADDR	Package	
417H	1047	IA32_MC5_MISC	Package	
418H	1048	IA32_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC6 reports MC errors from the integrated I/O module.
419H	1049	IA32_MC6_STATUS	Package	
41AH	1050	IA32_MC6_ADDR	Package	
41BH	1051	IA32_MC6_MISC	Package	
41CH	1052	IA32_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC7 reports MC errors from the M2M 0.
41DH	1053	IA32_MC7_STATUS	Package	
41EH	1054	IA32_MC7_ADDR	Package	
41FH	1055	IA32_MC7_MISC	Package	
420H	1056	IA32_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC8 reports MC errors from the M2M 1.
421H	1057	IA32_MC8_STATUS	Package	
422H	1058	IA32_MC8_ADDR	Package	
423H	1059	IA32_MC8_MISC	Package	
424H	1060	IA32_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 - MC11 report MC errors from the CHA
425H	1061	IA32_MC9_STATUS	Package	
426H	1062	IA32_MC9_ADDR	Package	
427H	1063	IA32_MC9_MISC	Package	

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
428H	1064	IA32_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 - MC11 report MC errors from the CHA.
429H	1065	IA32_MC10_STATUS	Package	
42AH	1066	IA32_MC10_ADDR	Package	
42BH	1067	IA32_MC10_MISC	Package	
42CH	1068	IA32_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC9 - MC11 report MC errors from the CHA.
42DH	1069	IA32_MC11_STATUS	Package	
42EH	1070	IA32_MC11_ADDR	Package	
42FH	1071	IA32_MC11_MISC	Package	
430H	1072	IA32_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC12 report MC errors from each channel of a link interconnect module.
431H	1073	IA32_MC12_STATUS	Package	
432H	1074	IA32_MC12_ADDR	Package	
433H	1075	IA32_MC12_MISC	Package	
434H	1076	IA32_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
435H	1077	IA32_MC13_STATUS	Package	
436H	1078	IA32_MC13_ADDR	Package	
437H	1079	IA32_MC13_MISC	Package	
438H	1080	IA32_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
439H	1081	IA32_MC14_STATUS	Package	
43AH	1082	IA32_MC14_ADDR	Package	
43BH	1083	IA32_MC14_MISC	Package	
43CH	1084	IA32_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
43DH	1085	IA32_MC15_STATUS	Package	
43EH	1086	IA32_MC15_ADDR	Package	
43FH	1087	IA32_MC15_MISC	Package	
440H	1088	IA32_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC13 through MC 18 report MC errors from the integrated memory controllers
441H	1089	IA32_MC16_STATUS	Package	
442H	1090	IA32_MC16_ADDR	Package	
443H	1091	IA32_MC16_MISC	Package	
444H	1092	IA32_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
445H	1093	IA32_MC17_STATUS	Package	
446H	1094	IA32_MC17_ADDR	Package	
447H	1095	IA32_MC17_MISC	Package	
448H	1096	IA32_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Banks MC13 through MC 18 report MC errors from the integrated memory controllers.
449H	1097	IA32_MC18_STATUS	Package	
44AH	1098	IA32_MC18_ADDR	Package	
44BH	1099	IA32_MC18_MISC	Package	

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
44CH	1100	IA32_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "IA32_MCi_MISC MSRs.". Bank MC19 reports MC errors from a link interconnect module.
44DH	1101	IA32_MC19_STATUS	Package	
44EH	1102	IA32_MC19_ADDR	Package	
44FH	1103	IA32_MC19_MISC	Package	
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 14.10.1, "RAPL Interfaces."
		7:4	Package	Reserved
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 14.10.1, "RAPL Interfaces."
		63:20		Reserved
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) Energy consumed by DRAM devices.
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
620H	1568	MSR UNCORE_RATIO_LIMIT	Package	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.
		63:15		Reserved
		14:8		MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.
		7		Reserved

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		6:0		MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.
639H	1593	MSR_PP0_ENERGY_STATUS	Package	Reserved (R/O) Reads return 0.
C8DH	3213	IA32_QM_EVTSEL	THREAD	Monitoring Event Select Register (R/W) If CPUID.(EAX=07H, ECX=0):EBX.RDT-M[bit 12] = 1.
		7:0		EventID (RW) 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 (RW)
		63:42		Reserved
C8FH	3215	IA32_PQR_ASSOC	THREAD	Resource Association Register (R/W)
		9:0		RMID
		31:10		Reserved
		51:32		COS (R/W)
		63: 52		Reserved
C90H	3216	IA32_L3_QOS_MASK_0	Package	L3 Class Of Service Mask - COS 0 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=0.
		0:19		CBM: Bit vector of available L3 ways for COS 0 enforcement.
		63:20		Reserved
C91H	3217	IA32_L3_QOS_MASK_1	Package	L3 Class Of Service Mask - COS 1 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=1.
		0:19		CBM: Bit vector of available L3 ways for COS 1 enforcement.
		63:20		Reserved
C92H	3218	IA32_L3_QOS_MASK_2	Package	L3 Class Of Service Mask - COS 2 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=2.
		0:19		CBM: Bit vector of available L3 ways for COS 2 enforcement.
		63:20		Reserved
C93H	3219	IA32_L3_QOS_MASK_3	Package	L3 Class Of Service Mask - COS 3 (R/W). If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=3.
		0:19		CBM: Bit vector of available L3 ways for COS 3 enforcement.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		63:20		Reserved
C94H	3220	IA32_L3_QOS_MASK_4	Package	L3 Class Of Service Mask - COS 4 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=4.
		0:19		CBM: Bit vector of available L3 ways for COS 4 enforcement.
		63:20		Reserved
C95H	3221	IA32_L3_QOS_MASK_5	Package	L3 Class Of Service Mask - COS 5 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=5.
		0:19		CBM: Bit vector of available L3 ways for COS 5 enforcement.
		63:20		Reserved
C96H	3222	IA32_L3_QOS_MASK_6	Package	L3 Class Of Service Mask - COS 6 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=6.
		0:19		CBM: Bit vector of available L3 ways for COS 6 enforcement.
		63:20		Reserved
C97H	3223	IA32_L3_QOS_MASK_7	Package	L3 Class Of Service Mask - COS 7 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=7.
		0:19		CBM: Bit vector of available L3 ways for COS 7 enforcement.
		63:20		Reserved
C98H	3224	IA32_L3_QOS_MASK_8	Package	L3 Class Of Service Mask - COS 8 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=8.
		0:19		CBM: Bit vector of available L3 ways for COS 8 enforcement.
		63:20		Reserved
C99H	3225	IA32_L3_QOS_MASK_9	Package	L3 Class Of Service Mask - COS 9 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=9.
		0:19		CBM: Bit vector of available L3 ways for COS 9 enforcement.
		63:20		Reserved
C9AH	3226	IA32_L3_QOS_MASK_10	Package	L3 Class Of Service Mask - COS 10 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=10.
		0:19		CBM: Bit vector of available L3 ways for COS 10 enforcement.
		63:20		Reserved
C9BH	3227	IA32_L3_QOS_MASK_11	Package	L3 Class Of Service Mask - COS 11 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=11.

Table 2-50. MSRs Supported by Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel 06_55H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		0:19		CBM: Bit vector of available L3 ways for COS 11 enforcement.
		63:20		Reserved
C9CH	3228	IA32_L3_QOS_MASK_12	Package	L3 Class Of Service Mask - COS 12 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=12.
		0:19		CBM: Bit vector of available L3 ways for COS 12 enforcement.
		63:20		Reserved
C9DH	3229	IA32_L3_QOS_MASK_13	Package	L3 Class Of Service Mask - COS 13 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=13.
		0:19		CBM: Bit vector of available L3 ways for COS 13 enforcement.
		63:20		Reserved
C9EH	3230	IA32_L3_QOS_MASK_14	Package	L3 Class Of Service Mask - COS 14 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=14.
		0:19		CBM: Bit vector of available L3 ways for COS 14 enforcement.
		63:20		Reserved
C9FH	3231	IA32_L3_QOS_MASK_15	Package	L3 Class Of Service Mask - COS 15 (R/W) If CPUID.(EAX=10H, ECX=1):EDX.COS_MAX[15:0] >=15.
		0:19		CBM: Bit vector of available L3 ways for COS 15 enforcement.
		63:20		Reserved

2.17.7 MSRs Specific to 3rd Generation Intel® Xeon® Processor Scalable Family based on Ice Lake Microarchitecture

The 3rd generation Intel® Xeon® Processor Scalable Family based on Ice Lake microarchitecture (CPUID DisplayFamily_DisplayModel signatures of 06_6AH and 06_6CH) support the MSRs listed in Table 2-51.

Table 2-51. MSRs Supported by 3rd Generation Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel Signatures of 06_6AH and 06_6CH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
612H	1554	MSR_PACKAGE_ENERGY_TIME_STATUS	Package	Package energy consumed by the entire CPU (R/W)
		31:0		Total amount of energy consumed since last reset.

Table 2-51. MSRs Supported by 3rd Generation Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel Signatures of 06_6AH and 06_6CH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
618H	1560	MSR_DRAM_POWER_LIMIT	Package	Allows software to set power limits for the DRAM domain and measurement attributes associated with each limit.
		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 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.
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
		31:0		Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).
		63:32		Reserved
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM Power Parameters (R/W)
		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
		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].

Table 2-51. MSRs Supported by 3rd Generation Intel® Xeon® Processor Scalable Family with DisplayFamily_DisplayModel Signatures of 06_6AH and 06_6CH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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 = PKG_MAX_WIN[54:53] y = PKG_MAX_WIN[52:48] The timing interval window is Floating Point number given by 1.x *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.
981H	2433	IA32_TME_CAPABILITY		See Table 2-2.
982H	2434	IA32_TME_ACTIVATE		See Table 2-2.
983H	2435	IA32_TME_EXCLUDE_MASK		See Table 2-2.
984H	2436	IA32_TME_EXCLUDE_BASE		See Table 2-2.

2.17.8 MSRs Introduced in Future Intel® Xeon® Processors

Table 2-52 lists additional MSRs for future Intel Xeon processors with a CPUID DisplayFamily_DisplayModel signature of 06_8FH. 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.

Table 2-52. Additional MSRs Supported by Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
CFH	207	IA32_CORE_CAPABILITY	Core	IA32 Core Capabilities Register (R/W) If CPUID.(EAX=07H, ECX=0):EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.
		0		Reserved: returns zero.

Table 2-52. Additional MSRs Supported by Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		1		Reserved: returns zero.
		2		INTEGRITY_CAPABILITIES When set to 1, the processor supports MSR_INTEGRITY_CAPABILITIES.
		3		RSM_IN_CPL0_ONLY Indicates that RSM will only be allowed in CPL0 and will #GP for all non-CPL0 privilege levels.
		4		UC_LOCK_DISABLE_SUPPORTED When set to 1, processor supports UC load lock disable feature.
		5		SPLIT_LOCK_DISABLE_SUPPORTED Indicates that there is core support for #AC on split lock detection feature.
		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.
2C2H	706	MSR_COPY_SCAN_HASHES	Die	COPY_SCAN_HASHES (W)
		63:0		SCAN_HASH_ADDR Contains the linear address of the SCAN Test HASH Binary loaded into memory.
2C3H	707	MSR_SCAN_HASHES_STATUS		SCAN_HASHES_STATUS (R/O)
		15:0	Die	CHUNK_SIZE Chunk size of the test in KB.
		23:16	Die	NUM_CHUNKS Total number of chunks.
		31:24		Reserved: all zeros.

Table 2-52. Additional MSRs Supported by Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		39:32	Thread	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.
		50:40		Reserved: set to all zeros.
		62:51	Die	MAX_CORE_LIMIT Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.
		63	Die	Valid Valid bit is set when COPY_SCAN_HASHES has completed successfully.
2C4H	708	MSR_AUTHENTICATE_AND_COPY_CHUNK	Die	AUTHENTICATE_AND_COPY_CHUNK (W)
		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.
2C5H	709	MSR_CHUNKS_AUTHENTICATION_STATUS		CHUNKS_AUTHENTICATION_STATUS (R/O)
		7:0	Die	VALID_CHUNKS Total number of Valid (authenticated) chunks.
		15:8	Die	TOTAL_CHUNKS Total number of chunks.
		31:16		Reserved: all zeros.

Table 2-52. Additional MSRs Supported by Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		39:32	Thread	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.
		63:40		Reserved: set to all zeros.
2C6H	710	MSR_ACTIVATE_SCAN	Thread	ACTIVATE_SCAN (W)
		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.
2C7H	711	MSR_SCAN_STATUS		SCAN_STATUS (R/O)
		7:0	Core	CHUNK_NUM SCAN Chunk that was reached.
		15:8	Core	CHUNK_STOP_INDEX Indicates what chunk index to stop at (inclusive). Maps to same field in WRMSR(ACTIVATE_SCAN).
		31:16		Reserved: return all zeros.

Table 2-52. Additional MSRs Supported by Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		39:32	Thread	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.
		61:40		Reserved: return all zeros.
		62	Core	SCAN_CONTROL_ERROR Scan-System-Controller malfunction.
		63	Core	SCAN_SIGNATURE_ERROR Core failed SCAN-SIGNATURE checking for this chunk.
2C8H	712	MSR_SCAN_MODULE_ID	Module	SCAN_MODULE_ID (R/O)
		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.
2C9H	713	MSR_LAST_SAF_WP	Core	LAST_SAF_WP (R/O)
		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.
2D9H	729	MSR_INTEGRITY_CAPABILITIES	Module	INTEGRITY_CAPABILITIES (R/O)
		0		STARTUP_SCAN_BIST When set, supports Intel In-field Scan.
		3:1		Reserved: return all zeros.

Table 2-52. Additional MSRs Supported by Future Intel® Xeon® Processors with a DisplayFamily_DisplayModel Signature of 06_8FH

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
		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.
410H	1040	IA32_MC4_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs" through Section 15.3.2.4, "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.
411H	1041	IA32_MC4_STATUS	Package	
412H	1042	IA32_MC4_ADDR	Package	
413H	1043	IA32_MC4_MISC	Package	

2.18 MSRS IN INTEL® XEON PHI™ PROCESSOR 3200/5200/7200 SERIES AND INTEL® XEON PHI™ PROCESSOR 7215/7285/7295 SERIES

Intel® Xeon Phi™ processor 3200, 5200, 7200 series, with CPUID DisplayFamily_DisplayModel signature 06_57H, supports the MSR interfaces listed in Table 2-53. These processors are based on the Knights Landing microarchitecture. Intel® Xeon Phi™ processor 7215, 7285, 7295 series, with CPUID DisplayFamily_DisplayModel signature 06_85H, supports the MSR interfaces listed in Table 2-53 and Table 2-54. These processors are based on the Knights Mill microarchitecture. Some MSRs are shared between a pair of processor cores, the scope is marked as module.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Module	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Module	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination." See Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Package	Platform ID (R) See Table 2-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 2-2.
34H	52	MSR_SMI_COUNT	Thread	SMI Counter (R/O)
		31:0		SMI Count (R/O)
		63:32		Reserved

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Thread	Control Features in Intel 64Processor (R/W) See Table 2-2.
		0		Lock (R/WL)
		1		Reserved
		2		Enable VMX outside SMX operation (R/WL)
3BH	59	IA32_TSC_ADJUST	THREAD	Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.
4EH	78	IA32_PPIN_CTL (MSR_PPIN_CTL)	Package	Protected Processor Inventory Number Enable Control (R/W)
		0		LockOut (R/WO) See Table 2-2.
		1		Enable_PPIN (R/W) See Table 2-2.
		63:2		Reserved
4FH	79	IA32_PPIN (MSR_PPIN)	Package	Protected Processor Inventory Number (R/O)
		63:0		Protected Processor Inventory Number (R/O) See Table 2-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	THREAD	BIOS Update Signature ID (RO) See Table 2-2.
C1H	193	IA32_PMC0	THREAD	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	THREAD	Performance Counter Register See Table 2-2.
CEH	206	MSR_PLATFORM_INFO	Package	Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org.
		7:0		Reserved
		15:8	Package	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved
		28	Package	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.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		29	Package	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.
		39:30		Reserved
		47:40	Package	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.
		63:48		Reserved
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Package	C-State Configuration Control (R/W)
		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 (RO) When set, locks bits [15:0] of this register for further writes until the next reset occurs.
		25		Reserved

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		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
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Tile	Power Management IO Capture Base (R/W)
		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
E7H	231	IA32_MPERF	Thread	Maximum Performance Frequency Clock Count (RW) See Table 2-2.
E8H	232	IA32_APERF	Thread	Actual Performance Frequency Clock Count (RW) See Table 2-2.
FEH	254	IA32_MTRRCAP	Core	Memory Type Range Register (R) See Table 2-2.
13CH	316	MSR_FEATURE_CONFIG	Core	AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.
		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
140H	320	MISC_FEATURE_ENABLES	Thread	MISC_FEATURE_ENABLES

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		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
174H	372	IA32_SYSENTER_CS	Thread	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 2-2.
179H	377	IA32_MCG_CAP	Thread	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Thread	See Table 2-2.
17DH	381	MSR_SMM_MCA_CAP	Thread	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		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
186H	390	IA32_PERFEVTSEL0	Thread	Performance Monitoring Event Select Register (R/W) See Table 2-2.
		7:0		Event Select
		15:8		UMask
		16		USR
		17		OS
		18		Edge

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		19		PC
		20		INT
		21		AnyThread
		22		EN
		23		INV
		31:24		CMASK
		63:32		Reserved
187H	391	IA32_PERFEVTSEL1	Thread	See Table 2-2.
198H	408	IA32_PERF_STATUS	Package	See Table 2-2.
199H	409	IA32_PERF_CTL	Thread	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	Clock Modulation (R/W) See Table 2-2.
19BH	411	IA32_THERM_INTERRUPT	Module	Thermal Interrupt Control (R/W) See Table 2-2.
19CH	412	IA32_THERM_STATUS	Module	Thermal Monitor Status (R/W) See Table 2-2.
		0		Thermal Status (RO)
		1		Thermal Status Log (R/WC0)
		2		PROTCHOT # or FORCEPR# Status (RO)
		3		PROTCHOT # or FORCEPR# Log (R/WC0)
		4		Critical Temperature Status (RO)
		5		Critical Temperature Status Log (R/WC0)
		6		Thermal Threshold #1 Status (RO)
		7		Thermal Threshold #1 Log (R/WC0)
		8		Thermal Threshold #2 Status (RO)
		9		Thermal Threshold #2 Log (R/WC0)
		10		Power Limitation Status (RO)
		11		Power Limitation Log (RWC0)
		15:12		Reserved
		22:16		Digital Readout (RO)
		26:23		Reserved
		30:27		Resolution in Degrees Celsius (RO)
		31		Reading Valid (RO)
		63:32		Reserved
1A0H	416	IA32_MISC_ENABLE	Thread	Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		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 (RO)
		12		Processor Event Based Sampling Unavailable (RO)
		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)
		37:35		Reserved
		38		Turbo Mode Disable (R/W)
		63:39		Reserved
1A2H	418	MSR_TEMPERATURE_TARGET	Package	Temperature Target
		15:0		Reserved
		23:16		Temperature Target (R)
		29:24		Target Offset (R/W)
		63:30		Reserved
1A4H	420	MSR_MISC_FEATURE_CONTROL		Miscellaneous Feature Control (R/W)
		0	Core	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher.
		1	Core	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher.
		63:2		Reserved
1A6H	422	MSR_OFFCORE_RSP_0	Shared	Offcore Response Event Select Register (R/W)
1A7H	423	MSR_OFFCORE_RSP_1	Shared	Offcore Response Event Select Register (R/W)
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode for Groups of Cores (RW)
		0		Reserved

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		7:1	Package	Maximum Number of Cores in Group 0 Number active processor cores which operates under the maximum ratio limit for group 0.
		15:8	Package	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.
		20:16	Package	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".
		23:21	Package	Group Ratio Delta for Group 1 An unsigned integer specifying the ratio decrement relative to the Max ratio limit to Group 0.
		28:24	Package	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".
		31:29	Package	Group Ratio Delta for Group 2 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 1.
		36:32	Package	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".
		39:37	Package	Group Ratio Delta for Group 3 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 2.
		44:40	Package	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".
		47:45	Package	Group Ratio Delta for Group 4 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 3.
		52:48	Package	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".

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		55:53	Package	Group Ratio Delta for Group 5 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 4.
		60:56	Package	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".
		63:61	Package	Group Ratio Delta for Group 6 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 5.
1B0H	432	IA32_ENERGY_PERF_BIAS	Thread	See Table 2-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 2-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 2-2.
1C8H	456	MSR_LBR_SELECT	Thread	Last Branch Record Filtering Select Register (R/W) See Section 17.9.2, "Filtering of Last Branch Records."
		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
1C9H	457	MSR_LASTBRANCH_TOS	Thread	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.
1D9H	473	IA32_DEBUGCTL	Thread	Debug Control (R/W)
		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.
		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

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
		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
1DDH	477	MSR_LER_FROM_LIP	Thread	Last Exception Record from Linear IP (R)
1DEH	478	MSR_LER_TO_LIP	Thread	Last Exception Record to Linear IP (R)
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 2-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 2-2.
200H	512	IA32_MTRR_PHYSBASE0	Core	See Table 2-2.
201H	513	IA32_MTRR_PHYSMASK0	Core	See Table 2-2.
202H	514	IA32_MTRR_PHYSBASE1	Core	See Table 2-2.
203H	515	IA32_MTRR_PHYSMASK1	Core	See Table 2-2.
204H	516	IA32_MTRR_PHYSBASE2	Core	See Table 2-2.
205H	517	IA32_MTRR_PHYSMASK2	Core	See Table 2-2.
206H	518	IA32_MTRR_PHYSBASE3	Core	See Table 2-2.
207H	519	IA32_MTRR_PHYSMASK3	Core	See Table 2-2.
208H	520	IA32_MTRR_PHYSBASE4	Core	See Table 2-2.
209H	521	IA32_MTRR_PHYSMASK4	Core	See Table 2-2.
20AH	522	IA32_MTRR_PHYSBASE5	Core	See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
20BH	523	IA32_MTRR_PHYSMASK5	Core	See Table 2-2.
20CH	524	IA32_MTRR_PHYSBASE6	Core	See Table 2-2.
20DH	525	IA32_MTRR_PHYSMASK6	Core	See Table 2-2.
20EH	526	IA32_MTRR_PHYSBASE7	Core	See Table 2-2.
20FH	527	IA32_MTRR_PHYSMASK7	Core	See Table 2-2.
250H	592	IA32_MTRR_FIX64K_00000	Core	See Table 2-2.
258H	600	IA32_MTRR_FIX16K_80000	Core	See Table 2-2.
259H	601	IA32_MTRR_FIX16K_A0000	Core	See Table 2-2.
268H	616	IA32_MTRR_FIX4K_C0000	Core	See Table 2-2.
269H	617	IA32_MTRR_FIX4K_C8000	Core	See Table 2-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Core	See Table 2-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Core	See Table 2-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Core	See Table 2-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Core	See Table 2-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Core	See Table 2-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Core	See Table 2-2.
277H	631	IA32_PAT	Core	See Table 2-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Core	Default Memory Types (R/W) See Table 2-2.
309H	777	IA32_FIXED_CTR0	Thread	Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.
30AH	778	IA32_FIXED_CTR1	Thread	Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.
30BH	779	IA32_FIXED_CTR2	Thread	Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.
345H	837	IA32_PERF_CAPABILITIES	Package	See Table 2-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Thread	Fixed-Function-Counter Control Register (R/W) See Table 2-2.
38EH	910	IA32_PERF_GLOBAL_STATUS	Thread	See Table 2-2.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 2-2.
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 2-2.
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	Thread	See Table 2-2.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		Package C3 Residency Counter (R/O)

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	
		63:0		Package C6 Residency Counter (R/O)
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	
		63:0		Package C7 Residency Counter (R/O)
3FCH	1020	MSR_MC0_RESIDENCY	Module	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		Module C0 Residency Counter (R/O)
3FDH	1021	MSR_MC6_RESIDENCY	Module	
		63:0		Module C6 Residency Counter (R/O)
3FFH	1023	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		CORE C6 Residency Counter (R/O)
400H	1024	IA32_MC0_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
404H	1028	IA32_MC1_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
410H	1040	IA32_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC4_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV 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.
414H	1044	IA32_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	IA32_MC5_STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
416H	1046	IA32_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
4C1H	1217	IA32_A_PMC0	Thread	See Table 2-2.
4C2H	1218	IA32_A_PMC1	Thread	See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
600H	1536	IA32_DS_AREA	Thread	DS Save Area (R/W) See Table 2-2.
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers Used in RAPL Interfaces (R/O)
		3:0	Package	Power Units See Section 14.10.1, "RAPL Interfaces."
		7:4	Package	Reserved
		12:8	Package	Energy Status Units Energy related information (in Joules) is based on the multiplier, 1/2^ESU; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).
		15:13	Package	Reserved
		19:16	Package	Time Units See Section 14.10.1, "RAPL Interfaces."
		63:20		Reserved
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.
		63:0		Package C2 Residency Counter (R/O)
610H	1552	MSR_PKG_POWER_LIMIT	Package	PKG RAPL Power Limit Control (R/W) See Section 14.10.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERGY_STATUS	Package	PKG Energy Status (R/O) See Section 14.10.3, "Package RAPL Domain."
613H	1555	MSR_PKG_PERF_STATUS	Package	PKG Perf Status (R/O) See Section 14.10.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	PKG RAPL Parameters (R/W) See Section 14.10.3, "Package RAPL Domain."
618H	1560	MSR_DRAM_POWER_LIMIT	Package	DRAM RAPL Power Limit Control (R/W) See Section 14.10.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENERGY_STATUS	Package	DRAM Energy Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF_STATUS	Package	DRAM Performance Throttling Status (R/O) See Section 14.10.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWER_INFO	Package	DRAM RAPL Parameters (R/W) See Section 14.10.5, "DRAM RAPL Domain."
638H	1592	MSR_PP0_POWER_LIMIT	Package	PP0 RAPL Power Limit Control (R/W) See Section 14.10.4, "PP0/PP1 RAPL Domains."
639H	1593	MSR_PP0_ENERGY_STATUS	Package	PP0 Energy Status (R/O) See Section 14.10.4, "PP0/PP1 RAPL Domains."

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
648H	1608	MSR_CONFIG_TDP_NOMINAL	Package	Base TDP Ratio (R/O) See Table 2-25.
649H	1609	MSR_CONFIG_TDP_LEVEL1	Package	ConfigTDP Level 1 ratio and power level (R/O) See Table 2-25.
64AH	1610	MSR_CONFIG_TDP_LEVEL2	Package	ConfigTDP Level 2 ratio and power level (R/O) See Table 2-25.
64BH	1611	MSR_CONFIG_TDP_CONTROL	Package	ConfigTDP Control (R/W) See Table 2-25.
64CH	1612	MSR_TURBO_ACTIVATION_RATIO	Package	ConfigTDP Control (R/W) See Table 2-25.
690H	1680	MSR_CORE_PERF_LIMIT_REASONS	Package	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)
		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
6E0H	1760	IA32_TSC_DEADLINE	Core	TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.
802H	2050	IA32_X2APIC_APICID	Thread	x2APIC ID Register (R/O)
803H	2051	IA32_X2APIC_VERSION	Thread	x2APIC Version Register (R/O)
808H	2056	IA32_X2APIC_TPR	Thread	x2APIC Task Priority Register (R/W)
80AH	2058	IA32_X2APIC_PPR	Thread	x2APIC Processor Priority Register (R/O)
80BH	2059	IA32_X2APIC_EOI	Thread	x2APIC EOI Register (W/O)
80DH	2061	IA32_X2APIC_LDR	Thread	x2APIC Logical Destination Register (R/O)
80FH	2063	IA32_X2APIC_SIVR	Thread	x2APIC Spurious Interrupt Vector Register (R/W)
810H	2064	IA32_X2APIC_ISR0	Thread	x2APIC In-Service Register Bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR1	Thread	x2APIC In-Service Register Bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR2	Thread	x2APIC In-Service Register Bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR3	Thread	x2APIC In-Service Register Bits [127:96] (R/O)
814H	2068	IA32_X2APIC_ISR4	Thread	x2APIC In-Service Register Bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service Register Bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service Register Bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service Register Bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Thread	x2APIC Trigger Mode Register Bits [31:0] (R/O)

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode Register Bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode Register Bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode Register Bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode Register Bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode Register Bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode Register Bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Thread	x2APIC Trigger Mode Register Bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Thread	x2APIC Interrupt Request Register Bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Thread	x2APIC Interrupt Request Register Bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Thread	x2APIC Interrupt Request Register Bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Thread	x2APIC Interrupt Request Register Bits [127:96] (R/O)
824H	2084	IA32_X2APIC_IRR4	Thread	x2APIC Interrupt Request Register Bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Thread	x2APIC Interrupt Request Register Bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Thread	x2APIC Interrupt Request Register Bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request Register Bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status Register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt Register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command Register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt Register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt Register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor Register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 Register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 Register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error Register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count Register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count Register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration Register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI Register (W/O)
C000_0080H		IA32_EFER	Thread	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	Thread	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	Thread	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	Thread	System Call Flag Mask (R/W) See Table 2-2.

Table 2-53. Selected MSRs Supported by Intel® Xeon Phi™ Processors with DisplayFamily_DisplayModel Signatures 06_57H and 06_85H

Register Address		Register Name / Bit Fields (Former MSR Name)	Scope	Bit Description
Hex	Dec			
C000_0100H		IA32_FS_BASE	Thread	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	Thread	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	Thread	Swap Target of BASE Address of GS (R/W) See Table 2-2.
C000_0103H		IA32_TSC_AUX	Thread	AUXILIARY TSC Signature (R/W) See Table 2-2

Table 2-54 lists model-specific registers that are supported by Intel® Xeon Phi™ processor 7215, 7285, 7295 series based on the Knights Mill microarchitecture.

Table 2-54. Additional MSRs Supported by Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with DisplayFamily_DisplayModel Signature 06_85H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
9BH	155	IA32_SMM_MONITOR_CTL	Core	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.
480H	1152	IA32_VMX_BASIC	Core	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2.
481H	1153	IA32_VMX_PINBASED_ CTLS	Core	Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2.
482H	1154	IA32_VMX_PROCBASED_ CTLS	Core	Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)
483H	1155	IA32_VMX_EXIT_CTLS	Core	Capability Reporting Register of VM-exit Controls (R/O) See Table 2-2.
484H	1156	IA32_VMX_ENTRY_CTLS	Core	Capability Reporting Register of VM-entry Controls (R/O) See Table 2-2.
485H	1157	IA32_VMX_MISC	Core	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2.
486H	1158	IA32_VMX_CR0_FIXED0	Core	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2.
487H	1159	IA32_VMX_CR0_FIXED1	Core	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2.

Table 2-54. Additional MSRs Supported by Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with DisplayFamily_DisplayModel Signature 06_85H

Register Address		Register Name / Bit Fields	Scope	Bit Description
Hex	Dec			
488H	1160	IA32_VMX_CR4_FIXED0	Core	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2.
489H	1161	IA32_VMX_CR4_FIXED1	Core	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2.
48AH	1162	IA32_VMX_VMCS_ENUM	Core	Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2.
48BH	1163	IA32_VMX_PROCBASED_ CTLS2	Core	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Table 2-2.
48CH	1164	IA32_VMX_EPT_VPID_ENUM	Core	Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.
48DH	1165	IA32_VMX_TRUE_PINBASE D_CTLS	Core	Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2.
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	Core	Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2.
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	Core	Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2.
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	Core	Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2.
491H	1169	IA32_VMX_FMFUNC	Core	Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.

2.19 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table 2-55 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 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
0H	0	IA32_P5_MC_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 2.23, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	0, 1, 2, 3, 4, 6	Shared	See Section 2.23, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_LINE_SIZE	3, 4, 6	Shared	See Section 8.10.5, "Monitor/Mwait Address Range Determination."
10H	16	IA32_TIME_STAMP_COUNTER	0, 1, 2, 3, 4, 6	Unique	Time Stamp Counter See Table 2-2.
					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.
17H	23	IA32_PLATFORM_ID	0, 1, 2, 3, 4, 6	Shared	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.
1BH	27	IA32_APIC_BASE	0, 1, 2, 3, 4, 6	Unique	APIC Location and Status (R/W) See Table 2-2. See Section 10.4.4, "Local APIC Status and Location."
2AH	42	MSR_EBC_HARD_POWERON	0, 1, 2, 3, 4, 6	Shared	Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.
		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.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		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.
		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
2BH	43	MSR_EBC_SOFT_POWERON	0, 1, 2, 3, 4, 6	Shared	Processor Soft Power-On Configuration (R/W) Enables and disables processor features.
		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.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		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
2CH	44	MSR_EBC_FREQUENCY_ID	2,3, 4, 6	Shared	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.
		15:0			Reserved
		18:16			Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: Encoding Scalable Bus Speed 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

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		31:24			Core Clock Frequency to System Bus Frequency Ratio (R) The processor core clock frequency to system bus frequency ratio observed at the de-assertion of the reset pin.
		63:25			Reserved
2CH	44	MSR_EBC_FREQUENCY_ID	0, 1	Shared	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.
		20:0			Reserved
		23:21			Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: Encoding Scalable Bus Speed 000B 100 MHz All others values reserved.
		63:24			Reserved
3AH	58	IA32_FEATURE_CONTROL	3, 4, 6	Unique	Control Features in IA-32 Processor (R/W) See Table 2-2. (If CPUID.01H:ECX.[bit 5])
79H	121	IA32_BIOS_UPDT_TRIG	0, 1, 2, 3, 4, 6	Shared	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	0, 1, 2, 3, 4, 6	Unique	BIOS Update Signature ID (R/W) See Table 2-2.
9BH	155	IA32_SMM_MONITOR_CTL	3, 4, 6	Unique	SMM Monitor Configuration (R/W) See Table 2-2.
FEH	254	IA32_MTRRCAP	0, 1, 2, 3, 4, 6	Unique	MTRR Information See Section 11.11.1, "MTRR Feature Identification.".
174H	372	IA32_SYSENTER_CS	0, 1, 2, 3, 4, 6	Unique	CS Register Target for CPL 0 Code (R/W) See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."
175H	373	IA32_SYSENTER_ESP	0, 1, 2, 3, 4, 6	Unique	Stack Pointer for CPL 0 Stack (R/W) See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
176H	374	IA32_SYSENTER_EIP	0, 1, 2, 3, 4, 6	Unique	CPL 0 Code Entry Point (R/W) See Table 2-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."
179H	377	IA32_MCG_CAP	0, 1, 2, 3, 4, 6	Unique	Machine Check Capabilities (R) See Table 2-2. See Section 15.3.1.1, "IA32_MCG_CAP MSR."
17AH	378	IA32_MCG_STATUS	0, 1, 2, 3, 4, 6	Unique	Machine Check Status (R) See Table 2-2. See Section 15.3.1.2, "IA32_MCG_STATUS MSR."
17BH	379	IA32_MCG_CTL			Machine Check Feature Enable (R/W) See Table 2-2. See Section 15.3.1.3, "IA32_MCG_CTL MSR."
180H	384	MSR_MCG_RAX	0, 1, 2, 3, 4, 6	Unique	Machine Check EAX/RAX Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
181H	385	MSR_MCG_RBX	0, 1, 2, 3, 4, 6	Unique	Machine Check EBX/RBX Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
182H	386	MSR_MCG_RCX	0, 1, 2, 3, 4, 6	Unique	Machine Check ECX/RCX Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
183H	387	MSR_MCG_RDX	0, 1, 2, 3, 4, 6	Unique	Machine Check EDX/RDX Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
184H	388	MSR_MCG_RSI	0, 1, 2, 3, 4, 6	Unique	Machine Check ESI/RSI Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
185H	389	MSR_MCG_RDI	0, 1, 2, 3, 4, 6	Unique	Machine Check EDI/RDI Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
186H	390	MSR_MCG_RBP	0, 1, 2, 3, 4, 6	Unique	Machine Check EBP/RBP Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
187H	391	MSR_MCG_RSP	0, 1, 2, 3, 4, 6	Unique	Machine Check ESP/RSP Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
188H	392	MSR_MCG_RFLAGS	0, 1, 2, 3, 4, 6	Unique	Machine Check EFLAGS/RFLAG Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
189H	393	MSR_MCG_RIP	0, 1, 2, 3, 4, 6	Unique	Machine Check EIP/RIP Save State See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
18AH	394	MSR_MCG_MISC	0, 1, 2, 3, 4, 6	Unique	Machine Check Miscellaneous See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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
18BH - 18FH	395-399	MSR_MCG_RESERVED1 - MSR_MCG_RESERVED5			Reserved

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
190H	400	MSR_MCG_R8	0, 1, 2, 3, 4, 6	Unique	Machine Check R8 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
191H	401	MSR_MCG_R9	0, 1, 2, 3, 4, 6	Unique	Machine Check R9D/R9 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
192H	402	MSR_MCG_R10	0, 1, 2, 3, 4, 6	Unique	Machine Check R10 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
193H	403	MSR_MCG_R11	0, 1, 2, 3, 4, 6	Unique	Machine Check R11 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
194H	404	MSR_MCG_R12	0, 1, 2, 3, 4, 6	Unique	Machine Check R12 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
195H	405	MSR_MCG_R13	0, 1, 2, 3, 4, 6	Unique	Machine Check R13 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		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.
196H	406	MSR_MCG_R14	0, 1, 2, 3, 4, 6	Unique	Machine Check R14 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
197H	407	MSR_MCG_R15	0, 1, 2, 3, 4, 6	Unique	Machine Check R15 See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		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.
198H	408	IA32_PERF_STATUS	3, 4, 6	Unique	See Table 2-2. See Section 14.1, "Enhanced Intel Speedstep® Technology."
199H	409	IA32_PERF_CTL	3, 4, 6	Unique	See Table 2-2. See Section 14.1, "Enhanced Intel Speedstep® Technology."
19AH	410	IA32_CLOCK_MODULATION	0, 1, 2, 3, 4, 6	Unique	Thermal Monitor Control (R/W) See Table 2-2. See Section 14.8.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_INTERRUPT	0, 1, 2, 3, 4, 6	Unique	Thermal Interrupt Control (R/W) See Section 14.8.2, "Thermal Monitor," and see Table 2-2.
19CH	412	IA32_THERM_STATUS	0, 1, 2, 3, 4, 6	Shared	Thermal Monitor Status (R/W) See Section 14.8.2, "Thermal Monitor," and see Table 2-2.
19DH	413	MSR_THERM2_CTL			Thermal Monitor 2 Control
			3,	Shared	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.
			4, 6	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.
1A0H	416	IA32_MISC_ENABLE	0, 1, 2, 3, 4, 6	Shared	Enable Miscellaneous Processor Features (R/W)

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		0			Fast-Strings Enable. See Table 2-2.
		1			Reserved
		2			x87 FPU Fopcode Compatibility Mode Enable
		3			Thermal Monitor 1 Enable See Section 14.8.2, "Thermal Monitor," and see 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 11.5.4, "Disabling and Enabling the L3 Cache."
		7			Performance Monitoring Available (R) See Table 2-2.
		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.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					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_UNAVILABLE) (R) See Table 2-2. When set, the processor does not support branch trace storage (BTS); when clear, BTS is supported.
		12			PEBS_UNAVILABLE: 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	3		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.
		17:14			Reserved
		18	3, 4, 6		ENABLE MONITOR FSM (R/W) See Table 2-2.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		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	3, 4, 6		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.
		23		Shared	xTPR Message Disable (R/W) See Table 2-2.
		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 11.5.6, "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		Unique	XD Bit Disable (R/W) See Table 2-2.
		63:35			Reserved
1A1H	417	MSR_PLATFORM_BRV	3, 4, 6	Shared	Platform Feature Requirements (R)
		17:0			Reserved

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		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
1D7H	471	MSR_LER_FROM_LIP	0, 1, 2, 3, 4, 6	Unique	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 17.13.3, "Last Exception Records."
		31:0			From Linear IP Linear address of the last branch instruction.
		63:32			Reserved
1D7H	471	63:0		Unique	From Linear IP Linear address of the last branch instruction (If IA-32e mode is active).
1D8H	472	MSR_LER_TO_LIP	0, 1, 2, 3, 4, 6	Unique	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 17.13.3, "Last Exception Records."
		31:0			From Linear IP Linear address of the target of the last branch instruction.
		63:32			Reserved
1D8H	472	63:0		Unique	From Linear IP Linear address of the target of the last branch instruction (If IA-32e mode is active).
1D9H	473	MSR_DEBUGCTLA	0, 1, 2, 3, 4, 6	Unique	Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 17.13.1, "MSR_DEBUGCTLA MSR."
1DAH	474	MSR_LASTBRANCH_TOS	0, 1, 2, 3, 4, 6	Unique	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 17.13.2, "LBR Stack for Processors Based on Intel NetBurst® Microarchitecture"; and addresses 1DBH-1DEH and 680H-68FH.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
1DBH	475	MSR_LASTBRANCH_0	0, 1, 2	Unique	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 17.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
1DCH	477	MSR_LASTBRANCH_1	0, 1, 2	Unique	Last Branch Record 1 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
1DDH	477	MSR_LASTBRANCH_2	0, 1, 2	Unique	Last Branch Record 2 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
1DEH	478	MSR_LASTBRANCH_3	0, 1, 2	Unique	Last Branch Record 3 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
200H	512	IA32_MTRR_PHYSBASE0	0, 1, 2, 3, 4, 6	Shared	Variable Range Base MTRR See Section 11.11.2.3, "Variable Range MTRRs."
201H	513	IA32_MTRR_PHYSMASK0	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
202H	514	IA32_MTRR_PHYSBASE1	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
203H	515	IA32_MTRR_PHYSMASK1	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
204H	516	IA32_MTRR_PHYSBASE2	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
205H	517	IA32_MTRR_PHYSMASK2	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs".
206H	518	IA32_MTRR_PHYSBASE3	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
207H	519	IA32_MTRR_PHYSMASK3	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
208H	520	IA32_MTRR_PHYSBASE4	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
209H	521	IA32_MTRR_PHYSMASK4	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
20AH	522	IA32_MTRR_PHYSBASE5	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20BH	523	IA32_MTRR_PHYSMASK5	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20CH	524	IA32_MTRR_PHYSBASE6	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20DH	525	IA32_MTRR_PHYSMASK6	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20EH	526	IA32_MTRR_PHYSBASE7	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
20FH	527	IA32_MTRR_PHYSMASK7	0, 1, 2, 3, 4, 6	Shared	Variable Range Mask MTRR See Section 11.11.2.3, "Variable Range MTRRs."
250H	592	IA32_MTRR_FIX64K_00000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
258H	600	IA32_MTRR_FIX16K_80000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
259H	601	IA32_MTRR_FIX16K_A0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
268H	616	IA32_MTRR_FIX4K_C0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
269H	617	IA32_MTRR_FIX4K_C8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs".
26AH	618	IA32_MTRR_FIX4K_D0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs".
26BH	619	IA32_MTRR_FIX4K_D8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26CH	620	IA32_MTRR_FIX4K_E0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26DH	621	IA32_MTRR_FIX4K_E8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26EH	622	IA32_MTRR_FIX4K_F0000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
26FH	623	IA32_MTRR_FIX4K_F8000	0, 1, 2, 3, 4, 6	Shared	Fixed Range MTRR See Section 11.11.2.2, "Fixed Range MTRRs."
277H	631	IA32_PAT	0, 1, 2, 3, 4, 6	Unique	Page Attribute Table See Section 11.11.2.2, "Fixed Range MTRRs."
2FFH	767	IA32_MTRR_DEF_TYPE	0, 1, 2, 3, 4, 6	Shared	Default Memory Types (R/W) See Table 2-2. See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
300H	768	MSR_BPU_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
301H	769	MSR_BPU_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
302H	770	MSR_BPU_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
303H	771	MSR_BPU_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
304H	772	MSR_MS_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
305H	773	MSR_MS_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
306H	774	MSR_MS_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
307H	775	MSR_MS_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
308H	776	MSR_FLAME_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
309H	777	MSR_FLAME_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30AH	778	MSR_FLAME_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30BH	779	MSR_FLAME_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30CH	780	MSR_IQ_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30DH	781	MSR_IQ_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30EH	782	MSR_IQ_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
30FH	783	MSR_IQ_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
310H	784	MSR_IQ_COUNTER4	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
311H	785	MSR_IQ_COUNTER5	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.2, "Performance Counters."
360H	864	MSR_BPU_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
361H	865	MSR_BPU_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
362H	866	MSR_BPU_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
363H	867	MSR_BPU_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
364H	868	MSR_MS_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
365H	869	MSR_MS_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
366H	870	MSR_MS_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
367H	871	MSR_MS_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
368H	872	MSR_FLAME_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
369H	873	MSR_FLAME_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36AH	874	MSR_FLAME_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36BH	875	MSR_FLAME_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36CH	876	MSR_IQ_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36DH	877	MSR_IQ_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36EH	878	MSR_IQ_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
36FH	879	MSR_IQ_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
370H	880	MSR_IQ_CCCR4	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
371H	881	MSR_IQ_CCCR5	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.3, "CCCR MSRs."
3A0H	928	MSR_BSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A1H	929	MSR_BSU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A2H	930	MSR_FSB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A3H	931	MSR_FSB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A4H	932	MSR_FIRM_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A5H	933	MSR_FIRM_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A6H	934	MSR_FLAME_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A7H	935	MSR_FLAME_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
3A8H	936	MSR_DAC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3A9H	937	MSR_DAC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3AAH	938	MSR_MOB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3ABH	939	MSR_MOB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3ACH	940	MSR_PMH_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3ADH	941	MSR_PMH_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3AEH	942	MSR_SAAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3AFH	943	MSR_SAAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B0H	944	MSR_U2L_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B1H	945	MSR_U2L_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B2H	946	MSR_BPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B3H	947	MSR_BPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B4H	948	MSR_IS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B5H	949	MSR_IS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B6H	950	MSR_ITLB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B7H	951	MSR_ITLB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B8H	952	MSR_CRU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3B9H	953	MSR_CRU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3BAH	954	MSR_IQ_ESCR0	0, 1, 2	Shared	See Section 19.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.
3BBH	955	MSR_IQ_ESCR1	0, 1, 2	Shared	See Section 19.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
3BCH	956	MSR_RAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3BDH	957	MSR_RAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3BEH	958	MSR_SSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C0H	960	MSR_MS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C1H	961	MSR_MS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C2H	962	MSR_TBPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C3H	963	MSR_TBPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C4H	964	MSR_TC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C5H	965	MSR_TC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C8H	968	MSR_IX_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3C9H	969	MSR_IX_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3CAH	970	MSR_ALF_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3CBH	971	MSR_ALF_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3CCH	972	MSR_CRU_ESCR2	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3CDH	973	MSR_CRU_ESCR3	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3E0H	992	MSR_CRU_ESCR4	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3E1H	993	MSR_CRU_ESCR5	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3F0H	1008	MSR_TC_PRECISE_EVENT	0, 1, 2, 3, 4, 6	Shared	See Section 19.6.3.1, "ESCR MSRs."
3F1H	1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	0, 1, 2, 3, 4, 6	Shared	Processor Event Based Sampling (PEBS) (R/W) Controls the enabling of processor event sampling and replay tagging.
		12:0			See https://perfmon-events.intel.com/.
		23:13			Reserved
		24			UOP Tag Enables replay tagging when set.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		25			ENABLE_PEBS_MY_THR (R/W) Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 19.6.4.3, "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 19.6.4.3, "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
3F2H	1010	MSR_PEBS_MATRIX_VERT	0, 1, 2, 3, 4, 6	Shared	See https://perfmon-events.intel.com/.
400H	1024	IA32_MC0_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	IA32_MC0_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC0_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
406H	1030	IA32_MC1_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV 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.
407H	1031	IA32_MC1_MISC		Shared	See Section 15.3.2.4, "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.
408H	1032	IA32_MC2_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR			See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV 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.
40BH	1035	IA32_MC2_MISC			See Section 15.3.2.4, "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.
40CH	1036	IA32_MC3_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC3_ADDR register is either not implemented or contains no address if the ADDRV 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.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
40FH	1039	IA32_MC3_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.4, "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.
410H	1040	IA32_MC4_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC4_ADDR			See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV 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.
413H	1043	IA32_MC4_MISC			See Section 15.3.2.4, "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.
480H	1152	IA32_VMX_BASIC	3, 4, 6	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTLS	3, 4, 6	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTLS	3, 4, 6	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and see Table 2-2.
483H	1155	IA32_VMX_EXIT_CTLS	3, 4, 6	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls," and see Table 2-2.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
484H	1156	IA32_VMX_ENTRY_CTLS	3, 4, 6	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls," and see Table 2-2.
485H	1157	IA32_VMX_MISC	3, 4, 6	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data," and see Table 2-2.
486H	1158	IA32_VMX_CR0_FIXED0	3, 4, 6	Unique	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and see Table 2-2.
487H	1159	IA32_VMX_CR0_FIXED1	3, 4, 6	Unique	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and see Table 2-2.
488H	1160	IA32_VMX_CR4_FIXED0	3, 4, 6	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and see Table 2-2.
489H	1161	IA32_VMX_CR4_FIXED1	3, 4, 6	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and see Table 2-2.
48AH	1162	IA32_VMX_VMCS_ENUM	3, 4, 6	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration," and see Table 2-2.
48BH	1163	IA32_VMX_PROCBASED_CTLS2	3, 4, 6	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and see Table 2-2.
600H	1536	IA32_DS_AREA	0, 1, 2, 3, 4, 6	Unique	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "Debug Store (DS) Mechanism."
680H	1664	MSR_LASTBRANCH_0_FROM_IP	3, 4, 6	Unique	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.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					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 17.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
681H	1665	MSR_LASTBRANCH_1_FROM_IP	3, 4, 6	Unique	Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 680H.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	3, 4, 6	Unique	Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 680H.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	3, 4, 6	Unique	Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 680H.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	3, 4, 6	Unique	Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 680H.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	3, 4, 6	Unique	Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 680H.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	3, 4, 6	Unique	Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 680H.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	3, 4, 6	Unique	Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 680H.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	3, 4, 6	Unique	Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 680H.
689H	1673	MSR_LASTBRANCH_9_FROM_IP	3, 4, 6	Unique	Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 680H.
68AH	1674	MSR_LASTBRANCH_10_FROM_IP	3, 4, 6	Unique	Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 680H.
68BH	1675	MSR_LASTBRANCH_11_FROM_IP	3, 4, 6	Unique	Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 680H.
68CH	1676	MSR_LASTBRANCH_12_FROM_IP	3, 4, 6	Unique	Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 680H.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
68DH	1677	MSR_LASTBRANCH_13_FROM_IP	3, 4, 6	Unique	Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 680H.
68EH	1678	MSR_LASTBRANCH_14_FROM_IP	3, 4, 6	Unique	Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 680H.
68FH	1679	MSR_LASTBRANCH_15_FROM_IP	3, 4, 6	Unique	Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 680H.
6C0H	1728	MSR_LASTBRANCH_0_TO_IP	3, 4, 6	Unique	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 17.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."
6C1H	1729	MSR_LASTBRANCH_1_TO_IP	3, 4, 6	Unique	Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 6C0H.
6C2H	1730	MSR_LASTBRANCH_2_TO_IP	3, 4, 6	Unique	Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 6C0H.
6C3H	1731	MSR_LASTBRANCH_3_TO_IP	3, 4, 6	Unique	Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 6C0H.
6C4H	1732	MSR_LASTBRANCH_4_TO_IP	3, 4, 6	Unique	Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 6C0H.
6C5H	1733	MSR_LASTBRANCH_5_TO_IP	3, 4, 6	Unique	Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 6C0H.
6C6H	1734	MSR_LASTBRANCH_6_TO_IP	3, 4, 6	Unique	Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 6C0H.
6C7H	1735	MSR_LASTBRANCH_7_TO_IP	3, 4, 6	Unique	Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 6C0H.
6C8H	1736	MSR_LASTBRANCH_8_TO_IP	3, 4, 6	Unique	Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 6C0H.

Table 2-55. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
6C9H	1737	MSR_LASTBRANCH_9_TO_IP	3, 4, 6	Unique	Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 6C0H.
6CAH	1738	MSR_LASTBRANCH_10_TO_IP	3, 4, 6	Unique	Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 6C0H.
6CBH	1739	MSR_LASTBRANCH_11_TO_IP	3, 4, 6	Unique	Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 6C0H.
6CCH	1740	MSR_LASTBRANCH_12_TO_IP	3, 4, 6	Unique	Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 6C0H.
6CDH	1741	MSR_LASTBRANCH_13_TO_IP	3, 4, 6	Unique	Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 6C0H.
6CEH	1742	MSR_LASTBRANCH_14_TO_IP	3, 4, 6	Unique	Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 6C0H.
6CFH	1743	MSR_LASTBRANCH_15_TO_IP	3, 4, 6	Unique	Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 6C0H.
C000_0080H		IA32_EFER	3, 4, 6	Unique	Extended Feature Enables See Table 2-2.
C000_0081H		IA32_STAR	3, 4, 6	Unique	System Call Target Address (R/W) See Table 2-2.
C000_0082H		IA32_LSTAR	3, 4, 6	Unique	IA-32e Mode System Call Target Address (R/W) See Table 2-2.
C000_0084H		IA32_FMASK	3, 4, 6	Unique	System Call Flag Mask (R/W) See Table 2-2.
C000_0100H		IA32_FS_BASE	3, 4, 6	Unique	Map of BASE Address of FS (R/W) See Table 2-2.
C000_0101H		IA32_GS_BASE	3, 4, 6	Unique	Map of BASE Address of GS (R/W) See Table 2-2.
C000_0102H		IA32_KERNEL_GS_BASE	3, 4, 6	Unique	Swap Target of BASE Address of GS (R/W) See Table 2-2.

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 2-56 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 of CPUID instruction (with EAX = 4 as input) 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 2-56. MSRs Unique to 64-bit Intel® Xeon® Processor MP with Up to an 8 MB L3 Cache

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique	Bit Description
107CCH		MSR_IFSB_BUSQ0	3, 4	Shared	IFSB BUSQ Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107CDH		MSR_IFSB_BUSQ1	3, 4	Shared	IFSB BUSQ Event Control and Counter Register (R/W)
107CEH		MSR_IFSB_SNPQ0	3, 4	Shared	IFSB SNPQ Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107CFH		MSR_IFSB_SNPQ1	3, 4	Shared	IFSB SNPQ Event Control and Counter Register (R/W)
107D0H		MSR_EFSB_DRDY0	3, 4	Shared	EFSB DRDY Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107D1H		MSR_EFSB_DRDY1	3, 4	Shared	EFSB DRDY Event Control and Counter Register (R/W)
107D2H		MSR_IFSB_CTL6	3, 4	Shared	IFSB Latency Event Control Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107D3H		MSR_IFSB_CNTR7	3, 4	Shared	IFSB Latency Event Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."

The MSRs listed in Table 2-57 apply to Intel® Xeon® Processor 7100 series. These processors can be detected by enumerating the deterministic cache parameter leaf of CPUID instruction (with EAX = 4 as input) 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 2-57 are shared between logical processors in the same core, but are replicated for each core.

Table 2-57. MSRs Unique to Intel® Xeon® Processor 7100 Series

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique	Bit Description
107CCH		MSR_EMON_L3_CTR_CTL0	6	Shared	GBUSQ Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."

Table 2-57. MSRs Unique to Intel® Xeon® Processor 7100 Series (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique	Bit Description
107CDH		MSR_EMON_L3_CTR_CTL1	6	Shared	GBUSQ Event Control and Counter Register (R/W)
107CEH		MSR_EMON_L3_CTR_CTL2	6	Shared	GSNPQ Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107CFH		MSR_EMON_L3_CTR_CTL3	6	Shared	GSNPQ Event Control and Counter Register (R/W)
107D0H		MSR_EMON_L3_CTR_CTL4	6	Shared	FSB Event Control and Counter Register (R/W) See Section 19.6.6, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107D1H		MSR_EMON_L3_CTR_CTL5	6	Shared	FSB Event Control and Counter Register (R/W)
107D2H		MSR_EMON_L3_CTR_CTL6	6	Shared	FSB Event Control and Counter Register (R/W)
107D3H		MSR_EMON_L3_CTR_CTL7	6	Shared	FSB Event Control and Counter Register (R/W)

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 2-58. 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 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	P5_MC_ADDR	Unique	See Section 2.23, "MSRs in Pentium Processors," and see Table 2-2.
1H	1	P5_MC_TYPE	Unique	See Section 2.23, "MSRs in Pentium Processors," and see Table 2-2.
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and see Table 2-2.
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
17H	23	IA32_PLATFORM_ID	Shared	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.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, "Local APIC Status and Location," and see Table 2-2.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
2AH	42	MSR_EBL_CR_POWERON	Shared	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/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
		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)

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_CONTROL	Unique	Control Features in IA-32 Processor (R/W) See Table 2-2.
40H	64	MSR_LASTBRANCH_0	Unique	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 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_LASTBRANCH_1	Unique	Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.
42H	66	MSR_LASTBRANCH_2	Unique	Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.
43H	67	MSR_LASTBRANCH_3	Unique	Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.
44H	68	MSR_LASTBRANCH_4	Unique	Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.
45H	69	MSR_LASTBRANCH_5	Unique	Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.
46H	70	MSR_LASTBRANCH_6	Unique	Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.
47H	71	MSR_LASTBRANCH_7	Unique	Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.
79H	121	IA32_BIOS_UPDT_TRIG	Unique	BIOS Update Trigger Register (W) See Table 2-2.
8BH	139	IA32_BIOS_SIGN_ID	Unique	BIOS Update Signature ID (RO) See Table 2-2.
C1H	193	IA32_PMC0	Unique	Performance Counter Register See Table 2-2.
C2H	194	IA32_PMC1	Unique	Performance Counter Register See Table 2-2.
CDH	205	MSR_FSB_FREQ	Shared	Scaleable Bus Speed (RO) This field indicates the scaleable bus clock speed:
		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

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
E7H	231	IA32_MPERF	Unique	Maximum Performance Frequency Clock Count (RW) See Table 2-2.
E8H	232	IA32_APERF	Unique	Actual Performance Frequency Clock Count (RW) See Table 2-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 2-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	Control Register 3 Used to configure the L2 Cache.
		0		L2 Hardware Enabled (RO) 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
		23		L2 Not Present (RO) 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved
174H	372	IA32_SYSENTER_CS	Unique	See Table 2-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 2-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 2-2.
179H	377	IA32_MCG_CAP	Unique	See Table 2-2.
17AH	378	IA32_MCG_STATUS	Unique	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.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		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
186H	390	IA32_PERFEVTSEL0	Unique	See Table 2-2.
187H	391	IA32_PERFEVTSEL1	Unique	See Table 2-2.
198H	408	IA32_PERF_STATUS	Shared	See Table 2-2.
199H	409	IA32_PERF_CTL	Unique	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	Clock Modulation (R/W) See Table 2-2.
19BH	411	IA32_THERM_INTERRUPT	Unique	Thermal Interrupt Control (R/W) See Table 2-2. See Section 14.8.2, "Thermal Monitor."
19CH	412	IA32_THERM_STATUS	Unique	Thermal Monitor Status (R/W) See Table 2-2. See Section 14.8.2, "Thermal Monitor".
19DH	413	MSR_THERM2_CTL	Unique	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
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	Unique	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.
		6:4		Reserved
		7	Shared	Performance Monitoring Available (R) See Table 2-2.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		9:8		Reserved
		10	Shared	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.
		11	Shared	Branch Trace Storage Unavailable (RO) See Table 2-2.
		12		Reserved
		13	Shared	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.
		15:14		Reserved
		16	Shared	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled
		18	Shared	ENABLE MONITOR FSM (R/W) See Table 2-2.
		19		Reserved
		22	Shared	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.
		33:23		Reserved
		34	Shared	XD Bit Disable (R/W) See Table 2-2.
		63:35		Reserved
1C9H	457	MSR_LASTBRANCH_TOS	Unique	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).

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
1D9H	473	IA32_DEBUGCTL	Unique	Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in Table 2-2.
1DDH	477	MSR_LER_FROM_LIP	Unique	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.
1DEH	478	MSR_LER_TO_LIP	Unique	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.
200H	512	MTRRphysBase0	Unique	Memory Type Range Registers
201H	513	MTRRphysMask0	Unique	Memory Type Range Registers
202H	514	MTRRphysBase1	Unique	Memory Type Range Registers
203H	515	MTRRphysMask1	Unique	Memory Type Range Registers
204H	516	MTRRphysBase2	Unique	Memory Type Range Registers
205H	517	MTRRphysMask2	Unique	Memory Type Range Registers
206H	518	MTRRphysBase3	Unique	Memory Type Range Registers
207H	519	MTRRphysMask3	Unique	Memory Type Range Registers
208H	520	MTRRphysBase4	Unique	Memory Type Range Registers
209H	521	MTRRphysMask4	Unique	Memory Type Range Registers
20AH	522	MTRRphysBase5	Unique	Memory Type Range Registers
20BH	523	MTRRphysMask5	Unique	Memory Type Range Registers
20CH	524	MTRRphysBase6	Unique	Memory Type Range Registers
20DH	525	MTRRphysMask6	Unique	Memory Type Range Registers
20EH	526	MTRRphysBase7	Unique	Memory Type Range Registers
20FH	527	MTRRphysMask7	Unique	Memory Type Range Registers
250H	592	MTRRfix64K_00000	Unique	Memory Type Range Registers
258H	600	MTRRfix16K_80000	Unique	Memory Type Range Registers
259H	601	MTRRfix16K_A0000	Unique	Memory Type Range Registers
268H	616	MTRRfix4K_C0000	Unique	Memory Type Range Registers
269H	617	MTRRfix4K_C8000	Unique	Memory Type Range Registers
26AH	618	MTRRfix4K_D0000	Unique	Memory Type Range Registers
26BH	619	MTRRfix4K_D8000	Unique	Memory Type Range Registers
26CH	620	MTRRfix4K_E0000	Unique	Memory Type Range Registers
26DH	621	MTRRfix4K_E8000	Unique	Memory Type Range Registers
26EH	622	MTRRfix4K_F0000	Unique	Memory Type Range Registers
26FH	623	MTRRfix4K_F8000	Unique	Memory Type Range Registers

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	Default Memory Types (R/W) See Table 2-2. See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
400H	1024	IA32_MC0_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV 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.
408H	1032	IA32_MC2_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV 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.
40CH	1036	MSR_MC4_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC4_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV 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.
410H	1040	IA32_MC3_CTL		See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC3_STATUS		See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	MSR_MC3_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV 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.

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
413H	1043	MSR_MC3_MISC	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.
414H	1044	MSR_MC5_CTL	Unique	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
415H	1045	MSR_MC5_STATUS	Unique	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.
416H	1046	MSR_MC5_ADDR	Unique	Machine Check Error Reporting Register - contains the address of the code or data memory location that produced the machine-check error if the ADDRV flag in the IA32_MCi_STATUS register is set.
417H	1047	MSR_MC5_MISC	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.
480H	1152	IA32_VMX_BASIC	Unique	Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information". (If CPUID.01H:ECX.[bit 5])
481H	1153	IA32_VMX_PINBASED_CTLS	Unique	Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls". (If CPUID.01H:ECX.[bit 5])
482H	1154	IA32_VMX_PROCBASED_CTLS	Unique	Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls". (If CPUID.01H:ECX.[bit 5])
483H	1155	IA32_VMX_EXIT_CTLS	Unique	Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls". (If CPUID.01H:ECX.[bit 5])
484H	1156	IA32_VMX_ENTRY_CTLS	Unique	Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls". (If CPUID.01H:ECX.[bit 5])
485H	1157	IA32_VMX_MISC	Unique	Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data". (If CPUID.01H:ECX.[bit 5])
486H	1158	IA32_VMX_CR0_FIXED0	Unique	Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0". (If CPUID.01H:ECX.[bit 5])

Table 2-58. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
487H	1159	IA32_VMX_CR0_FIXED1	Unique	Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0". (If CPUID.01H:ECX.[bit 5])
488H	1160	IA32_VMX_CR4_FIXED0	Unique	Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4". (If CPUID.01H:ECX.[bit 5])
489H	1161	IA32_VMX_CR4_FIXED1	Unique	Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4". (If CPUID.01H:ECX.[bit 5])
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration". (If CPUID.01H:ECX.[bit 5])
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Unique	Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls". (If CPUID.01H:ECX.[bit 5] and IA32_VMX_PROCBASED_CTLS[bit 63])
600H	1536	IA32_DS_AREA	Unique	DS Save Area (R/W) See Table 2-2. See Section 19.6.3.4, "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
C000_ 0080H		IA32_EFER	Unique	See Table 2-2.
		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 2-59. MSRs in Pentium M Processors

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 2.23, "MSRs in Pentium Processors."

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
1H	1	P5_MC_TYPE	See Section 2.23, "MSRs in Pentium Processors."
10H	16	IA32_TIME_STAMP_COUNTER	See Section 17.17, "Time-Stamp Counter," and see Table 2-2.
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.
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.
		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

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		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)
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 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)".
41H	65	MSR_LASTBRANCH_1	Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.
42H	66	MSR_LASTBRANCH_2	Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.
43H	67	MSR_LASTBRANCH_3	Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.
44H	68	MSR_LASTBRANCH_4	Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.
45H	69	MSR_LASTBRANCH_5	Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.
46H	70	MSR_LASTBRANCH_6	Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.
47H	71	MSR_LASTBRANCH_7	Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.
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
11EH	281	MSR_BBL_CR_CTL3	Control register 3 Used to configure the L2 Cache.

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		0	L2 Hardware Enabled (RO) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.
		4:1	Reserved
		5	ECC Check Enable (RO) 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 (RO) 0 = L2 Present 1 = L2 Not Present
		63:24	Reserved
179H	377	IA32_MCG_CAP	Read-only register that provides information about the machine-check architecture of the processor.
		7:0	Count (RO) Indicates the number of hardware unit error reporting banks available in the processor.
		8	IA32_MCG_CTL Present (RO) 1 = Indicates that the processor implements the MSR_MCG_CTL register found at MSR 17BH. 0 = Not supported.
		63:9	Reserved
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.

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		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
198H	408	IA32_PERF_STATUS	See Table 2-2.
199H	409	IA32_PERF_CTL	See Table 2-2.
19AH	410	IA32_CLOCK_MODULATION	Clock Modulation (R/W). See Table 2-2. See Section 14.8.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_INTERRUPT	Thermal Interrupt Control (R/W) See Table 2-2. See Section 14.8.2, "Thermal Monitor."
19CH	412	IA32_THERM_STATUS	Thermal Monitor Status (R/W) See Table 2-2. See Section 14.8.2, "Thermal Monitor."
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
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

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		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.
		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 (RO) 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported
		12	Processor Event Based Sampling Unavailable (RO) 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

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
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 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)".
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 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
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 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)" and Section 17.16.2, "Last Branch and Last Exception MSRs."
1DEH	478	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. See Section 17.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)" and Section 17.16.2, "Last Branch and Last Exception MSRs."
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 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
400H	1024	IA32_MC0_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	See Section 14.3.2.3., "IA32_MCi_ADDR MSRs". The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV 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.
408H	1032	IA32_MC2_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	See Chapter 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 2-59. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
40AH	1034	IA32_MC2_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV 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.
40CH	1036	MSR_MC4_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC4_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV 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.
410H	1040	MSR_MC3_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC3_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	MSR_MC3_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV 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.
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 19.6.3.4, "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 2-60. MSRs in the P6 Family Processors

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 2.23, "MSRs in Pentium Processors."
1H	1	P5_MC_TYPE	See Section 2.23, "MSRs in Pentium Processors."
10H	16	TSC	See Section 17.17, "Time-Stamp Counter."

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
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
		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
1BH	27	APIC_BASE	Section 10.4.4, "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
2AH	42	EBL_CR_POWERON	Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (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

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		4	BERR# Enable for Initiator Bus Requests (R/W) 1 = Enabled 0 = Disabled
		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)

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		27	Clock Frequency Ratio
		63:28	Reserved[1]
33H	51	MSR_TEST_CTRL	Test Control Register
		29:0	Reserved
		30	Streaming Buffer Disable
		31	Disable LOCK# Assertion for split locked access.
79H	121	BIOS_UPDT_TRIG	BIOS Update Trigger Register.
88H	136	BBL_CR_D0[63:0]	Chunk 0 data register D[63:0]: used to write to and read from the L2
89H	137	BBL_CR_D1[63:0]	Chunk 1 data register D[63:0]: used to write to and read from the L2
8AH	138	BBL_CR_D2[63:0]	Chunk 2 data register D[63:0]: used to write to and read from the L2
8BH	139	BIOS_SIGN/BBL_CR_D3[63:0]	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.
C1H	193	PerfCtr0 (PERFCTR0)	Performance Counter Register See Table 2-2.
C2H	194	PerfCtr1 (PERFCTR1)	Performance Counter Register See Table 2-2.
FEH	254	MTRRcap	Memory Type Range Registers
116H	278	BBL_CR_ADDR [63:0] BBL_CR_ADDR [63:32] BBL_CR_ADDR [31:3] BBL_CR_ADDR [2:0]	Address register: used to send specified address (A31-A3) to L2 during cache initialization accesses. Reserved, Address bits [35:3] Reserved Set to 0.
118H	280	BBL_CR_DECC[63:0]	Data ECC register D[7:0]: used to write ECC and read ECC to/from L2
119H	281	BBL_CR_CTL BL_CR_CTL[63:22] BBL_CR_CTL[21]	Control register: used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response Reserved Processor number[2] Disable = 1 Enable = 0 Reserved

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		BBL_CR_CTL[20:19]	User supplied ECC
		BBL_CR_CTL[18]	Reserved
		BBL_CR_CTL[17]	L2 Hit
		BBL_CR_CTL[16]	Reserved
		BBL_CR_CTL[15:14]	State from L2
		BBL_CR_CTL[13:12]	Modified - 11,Exclusive - 10, Shared - 01, Invalid - 00
		BBL_CR_CTL[11:10]	Way from L2 Way 0 - 00, Way 1 - 01, Way 2 - 10, Way 3 - 11
		BBL_CR_CTL[9:8]	Way to L2 Reserved
		BBL_CR_CTL[7]	State to L2
		BBL_CR_CTL[6:5]	
		BBL_CR_CTL[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)
11AH	282	BBL_CR_TRIG	Trigger register: used to initiate a cache configuration accesses access, Write only with Data = 0.
11BH	283	BBL_CR_BUSY	Busy register: indicates when a cache configuration accesses L2 command is in progress. D[0] = 1 = BUSY
11EH	286	BBL_CR_CTL3	Control register 3: used to configure the L2 Cache
		BBL_CR_CTL3[63:26]	Reserved
		BBL_CR_CTL3[25]	Cache bus fraction (read only)
		BBL_CR_CTL3[24]	Reserved
		BBL_CR_CTL3[23]	L2 Hardware Disable (read only)
		BBL_CR_CTL3[22:20]	L2 Physical Address Range support
		111	64GBytes
		110	32GBytes
		101	16GBytes
		100	8GBytes
		011	4GBytes
		010	2GBytes
		001	1GBytes
		000	512MBytes
		BBL_CR_CTL3[19]	Reserved
		BBL_CR_CTL3[18]	Cache State error checking enable (read/write)

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		BBL_CR_CTL3[17:13 00001 00010 00100 01000 10000 BBL_CR_CTL3[12:11] BBL_CR_CTL3[10:9] 00 01 10 11 BBL_CR_CTL3[8] BBL_CR_CTL3[7] BBL_CR_CTL3[6] BBL_CR_CTL3[5] BBL_CR_CTL3[4:1] BBL_CR_CTL3[0]	Cache size per bank (read/write) 256KBytes 512KBytes 1MByte 2MByte 4MBytes Number of L2 banks (read only) L2 Associativity (read only) Direct Mapped 2 Way 4 Way Reserved L2 Enabled (read/write) CRTN Parity Check Enable (read/write) Address Parity Check Enable (read/write) ECC Check Enable (read/write) L2 Cache Latency (read/write) L2 Configured (read/write)
174H	372	SYSENTER_CS_MSR	CS register target for CPL 0 code
175H	373	SYSENTER_ESP_MSR	Stack pointer for CPL 0 stack
176H	374	SYSENTER_EIP_MSR	CPL 0 code entry point
179H	377	MCG_CAP	Machine Check Global Control Register
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.
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).
186H	390	PerfEvtSel0 (EVNTSEL0)	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.

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		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
		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)
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.
		18	E Occurrence/Duration Mode Select. 1 = Occurrence 0 = Duration
		19	PC Enabled the signaling of performance counter overflow via BP0 pin.

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
		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)
1D9H	473	DEBUGCTLMSR	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
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.
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.
1DDH	477	LASTINTFROMIP	Last INT from IP
1DEH	478	LASTINTTOIP	Last INT to IP
200H	512	MTRRphysBase0	Memory Type Range Registers
201H	513	MTRRphysMask0	Memory Type Range Registers
202H	514	MTRRphysBase1	Memory Type Range Registers
203H	515	MTRRphysMask1	Memory Type Range Registers
204H	516	MTRRphysBase2	Memory Type Range Registers
205H	517	MTRRphysMask2	Memory Type Range Registers
206H	518	MTRRphysBase3	Memory Type Range Registers
207H	519	MTRRphysMask3	Memory Type Range Registers
208H	520	MTRRphysBase4	Memory Type Range Registers
209H	521	MTRRphysMask4	Memory Type Range Registers
20AH	522	MTRRphysBase5	Memory Type Range Registers

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
20BH	523	MTRRphysMask5	Memory Type Range Registers
20CH	524	MTRRphysBase6	Memory Type Range Registers
20DH	525	MTRRphysMask6	Memory Type Range Registers
20EH	526	MTRRphysBase7	Memory Type Range Registers
20FH	527	MTRRphysMask7	Memory Type Range Registers
250H	592	MTRRfix64K_00000	Memory Type Range Registers
258H	600	MTRRfix16K_80000	Memory Type Range Registers
259H	601	MTRRfix16K_A0000	Memory Type Range Registers
268H	616	MTRRfix4K_C0000	Memory Type Range Registers
269H	617	MTRRfix4K_C8000	Memory Type Range Registers
26AH	618	MTRRfix4K_D0000	Memory Type Range Registers
26BH	619	MTRRfix4K_D8000	Memory Type Range Registers
26CH	620	MTRRfix4K_E0000	Memory Type Range Registers
26DH	621	MTRRfix4K_E8000	Memory Type Range Registers
26EH	622	MTRRfix4K_F0000	Memory Type Range Registers
26FH	623	MTRRfix4K_F8000	Memory Type Range Registers
2FFH	767	MTRRdefType	Memory Type Range Registers
		2:0	Default memory type
		10	Fixed MTRR enable
		11	MTRR Enable
400H	1024	MC0_CTL	Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).
401H	1025	MC0_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 MC0_STATUS only, this bit is hardcoded to 1.)
		61	MC_STATUS_UC
		62	MC_STATUS_O
		63	MC_STATUS_V
402H	1026	MC0_ADDR	
403H	1027	MC0_MISC	Defined in MCA architecture but not implemented in the P6 family processors.

Table 2-60. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name / Bit Fields	Bit Description
Hex	Dec		
404H	1028	MC1_CTL	
405H	1029	MC1_STATUS	Bit definitions same as MC0_STATUS.
406H	1030	MC1_ADDR	
407H	1031	MC1_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
408H	1032	MC2_CTL	
409H	1033	MC2_STATUS	Bit definitions same as MC0_STATUS.
40AH	1034	MC2_ADDR	
40BH	1035	MC2_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
40CH	1036	MC4_CTL	
40DH	1037	MC4_STATUS	Bit definitions same as MC0_STATUS, except bits 0, 4, 57, and 61 are hardcoded to 1.
40EH	1038	MC4_ADDR	Defined in MCA architecture but not implemented in P6 Family processors.
40FH	1039	MC4_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
410H	1040	MC3_CTL	
411H	1041	MC3_STATUS	Bit definitions same as MC0_STATUS.
412H	1042	MC3_ADDR	
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 2-61. MSRs in the Pentium Processor

Register Address		Register Name	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 15.10.2, "Pentium Processor Machine-Check Exception Handling."
1H	1	P5_MC_TYPE	See Section 15.10.2, "Pentium Processor Machine-Check Exception Handling."
10H	16	TSC	See Section 17.17, "Time-Stamp Counter."
11H	17	CESR	See Section 19.6.9.1, "Control and Event Select Register (CESR)."
12H	18	CTR0	Section 19.6.9.3, "Events Counted."
13H	19	CTR1	Section 19.6.9.3, "Events Counted."

2.24 MSR INDEX

MSRs of recent processors are indexed here for convenience. IA32 MSRs are excluded from this index.

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_B0_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
MSR_B0_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
MSR_B0_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
MSR_B0_PMON_MASK	
06_2EH	See Table 2-17
MSR_B0_PMON_MATCH	
06_2EH	See Table 2-17
MSR_B1_PMON_BOX_CTRL	
06_2EH	See Table 2-17
MSR_B1_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_B1_PMON_BOX_STATUS	
06_2EH	See Table 2-17
MSR_B1_PMON_CTR0	
06_2EH	See Table 2-17
MSR_B1_PMON_CTR1	
06_2EH	See Table 2-17
MSR_B1_PMON_CTR2	
06_2EH	See Table 2-17
MSR_B1_PMON_CTR3	
06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
MSR_B1_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
MSR_B1_PMON_MASK	
06_2EH	See Table 2-17
MSR_B1_PMON_MATCH	
06_2EH	See Table 2-17
MSR_BBL_CR_CTL	
06_09H	See Table 2-59
MSR_BBL_CR_CTL3	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_BIOS_DEBUG	
06_8CH, 06_8DH	See Table 2-45
MSR_BIOS_DONE	
06_7DH, 06_7EH	See Table 2-44
MSR_BIOS_MCU_ERRORCODE	
06_7DH, 06_7EH	See Table 2-44
06_8CH, 06_8DH	See Table 2-45
MSR_BPU_CCCR0	
0FH	See Table 2-55
MSR_BPU_CCCR1	
0FH	See Table 2-55
MSR_BPU_CCCR2	
0FH	See Table 2-55
MSR_BPU_CCCR3	
0FH	See Table 2-55
MSR_BPU_COUNTER0	
0FH	See Table 2-55
MSR_BPU_COUNTER1	
0FH	See Table 2-55
MSR_BPU_COUNTER2	
0FH	See Table 2-55
MSR_BPU_COUNTER3	
0FH	See Table 2-55
MSR_BPU_ESCR0	
0FH	See Table 2-55
MSR_BPU_ESCR1	
0FH	See Table 2-55
MSR_BR_DETECT_COUNTER_CONFIG_i	
06_66H	See Table 2-42
MSR_BR_DETECT_CTRL	
06_66H	See Table 2-42
MSR_BR_DETECT_STATUS	
06_66H	See Table 2-42
MSR_BSU_ESCR0	
0FH	See Table 2-55
MSR_BSU_ESCR1	
0FH	See Table 2-55
MSR_C0_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3FH	See Table 2-33
MSR_C0_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C0_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C0_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C0_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C0_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C0_PMON_CTR0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C0_PMON_CTR1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C0_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C0_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C0_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C0_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C0_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C0_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C0_PMON_CTR1	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C0_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C0_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C0_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C0_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C0_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C1_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C1_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C1_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C1_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C1_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C1_PMON_CTR0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_CTR1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3FH	See Table 2-33
MSR_C1_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C1_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C1_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C1_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C1_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C10_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C10_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C10_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C11_PMON_BOX_FILTER	
06_3EH	See Table 2-28

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C11_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C11_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C12_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C12_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C12_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C13_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C13_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C13_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C14_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C14_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C14_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C15_PMON_BOX_CTL	
06_3FH	See Table 2-33
MSR_C15_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C15_PMON_BOX_FILTER1	
06_3FH	See Table 2-33
MSR_C15_PMON_BOX_STATUS	
06_3FH	See Table 2-33
MSR_C15_PMON_CTR0	
06_3FH	See Table 2-33
MSR_C15_PMON_CTR1	
06_3FH	See Table 2-33
MSR_C15_PMON_CTR2	
06_3FH	See Table 2-33
MSR_C15_PMON_CTR3	
06_3FH	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C15_PMON_EVNTSEL0	
06_3FH	See Table 2-33
MSR_C15_PMON_EVNTSEL1	
06_3FH	See Table 2-33
MSR_C15_PMON_EVNTSEL2	
06_3FH	See Table 2-33
MSR_C15_PMON_EVNTSEL3	
06_3FH	See Table 2-33
MSR_C16_PMON_BOX_CTL	
06_3FH	See Table 2-33
MSR_C16_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C16_PMON_BOX_FILTER1	
06_3FH	See Table 2-33
MSR_C16_PMON_BOX_STATUS	
06_3FH	See Table 2-33
MSR_C16_PMON_CTR0	
06_3FH	See Table 2-33
MSR_C16_PMON_CTR3	
06_3FH	See Table 2-33
MSR_C16_PMON_CTR2	
06_3FH	See Table 2-33
MSR_C16_PMON_CTR3	
06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSEL0	
06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSEL1	
06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSEL2	
06_3FH	See Table 2-33
MSR_C16_PMON_EVNTSEL3	
06_3FH	See Table 2-33
MSR_C17_PMON_BOX_CTL	
06_3FH	See Table 2-33
MSR_C17_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C17_PMON_BOX_FILTER1	
06_3FH	See Table 2-33
MSR_C17_PMON_BOX_STATUS	
06_3FH	See Table 2-33
MSR_C17_PMON_CTR0	
06_3FH	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C17_PMON_CTR1	
06_3FH	See Table 2-33
MSR_C17_PMON_CTR2	
06_3FH	See Table 2-33
MSR_C17_PMON_CTR3	
06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSEL0	
06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSEL1	
06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSEL2	
06_3FH	See Table 2-33
MSR_C17_PMON_EVNTSEL3	
06_3FH	See Table 2-33
MSR_C2_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C2_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C2_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C2_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C2_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C2_PMON_CTR0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_CTR1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C2_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C2_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C2_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C2_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C2_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C3_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C3_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C3_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C3_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C3_PMON_BOX_STATUS	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C3_PMON_CTR0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTR1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C3_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C3_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C3_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C3_PMON_EVNT_SEL5	
06_2EH	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C4_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C4_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C4_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C4_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C4_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C4_PMON_CTR0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTR1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C4_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C4_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL1	
06_2EH	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C4_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C4_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C5_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C5_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C5_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C5_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C5_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C5_PMON_CTR0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_CTR1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C5_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C5_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C5_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C5_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C5_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C6_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C6_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C6_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C6_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C6_PMON_BOX_STATUS	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C6_PMON_CTR0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTR1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C6_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C6_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C6_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C6_PMON_EVNT_SEL5	
06_2EH	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C7_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_BOX_FILTER	
06_2DH	See Table 2-24
MSR_C7_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C7_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C7_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_C7_PMON_BOX_STATUS	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_C7_PMON_CTR0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTR1	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTR2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTR3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_CTR4	
06_2EH	See Table 2-17
MSR_C7_PMON_CTR5	
06_2EH	See Table 2-17
MSR_C7_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL1	
06_2EH	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_C7_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_C7_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_C8_PMON_BOX_CTRL	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C8_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C8_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_BOX_OVF_CTRL	
06_2FH	See Table 2-19
MSR_C8_PMON_BOX_STATUS	
06_2FH	See Table 2-19
06_3FH	See Table 2-33
MSR_C8_PMON_CTR0	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_CTR1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_CTR2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_C8_PMON_CTR3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_CTR4	
06_2FH	See Table 2-19
MSR_C8_PMON_CTR5	
06_2FH	See Table 2-19
MSR_C8_PMON_EVNT_SEL0	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C8_PMON_EVNT_SEL4	
06_2FH	See Table 2-19
MSR_C8_PMON_EVNT_SEL5	
06_2FH	See Table 2-19
MSR_C9_PMON_BOX_CTRL	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_BOX_FILTER	
06_3EH	See Table 2-28
MSR_C9_PMON_BOX_FILTER0	
06_3FH	See Table 2-33
MSR_C9_PMON_BOX_FILTER1	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_BOX_OVF_CTRL	
06_2FH	See Table 2-19
MSR_C9_PMON_BOX_STATUS	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2FH	See Table 2-19
06_3FH	See Table 2-33
MSR_C9_PMON_CTR0	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTR1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTR2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTR3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_CTR4	
06_2FH	See Table 2-19
MSR_C9_PMON_CTR5	
06_2FH	See Table 2-19
MSR_C9_PMON_EVNT_SEL0	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVNT_SEL1	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVNT_SEL2	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVNT_SEL3	
06_2FH	See Table 2-19
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_C9_PMON_EVNT_SEL4	
06_2FH	See Table 2-19
MSR_C9_PMON_EVNT_SEL5	
06_2FH	See Table 2-19

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_CC6_DEMOTION_POLICY_CONFIG	
06_37H	See Table 2-9
MSR_CONFIG_TDP_CONTROL	
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_CONFIG_TDP_LEVEL1	
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H.	See Table 2-53
MSR_CONFIG_TDP_LEVEL2	
06_3AH.	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_CONFIG_TDP_NOMINAL	
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_CORE_C1_RESIDENCY	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.	See Table 2-6
06_66H	See Table 2-42
MSR_CORE_C3_RESIDENCY	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
MSR_CORE_C6_RESIDENCY	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H, 06_85H.	See Table 2-53
MSR_CORE_C7_RESIDENCY	
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
MSR_CORE_GFXE_OVERLAP_C0	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_CORE_HDC_RESIDENCY	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_CORE_PERF_LIMIT_REASONS	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H	See Table 2-30
06_3F	See Table 2-32

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_56H, 06_4FH	See Table 2-36
06_57H, 06_85H	See Table 2-53
MSR_CORE_THREAD_COUNT	
06_3FH	See Table 2-32
MSR_CRASHLOG_CONTROL	
06_7DH, 06_7EH	See Table 2-44
MSR_CRU_ESCR0	
0FH	See Table 2-55
MSR_CRU_ESCR1	
0FH	See Table 2-55
MSR_CRU_ESCR2	
0FH	See Table 2-55
MSR_CRU_ESCR3	
0FH	See Table 2-55
MSR_CRU_ESCR4	
0FH	See Table 2-55
MSR_CRU_ESCR5	
0FH	See Table 2-55
MSR_DAC_ESCR0	
0FH	See Table 2-55
MSR_DAC_ESCR1	
0FH	See Table 2-55
MSR_DRAM_ENERGY_ STATUS	
06_5CH, 06_7AH	See Table 2-12
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-29
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H	See Table 2-53
MSR_DRAM_PERF_STATUS	
06_5CH, 06_7AH	See Table 2-12
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-29
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H	See Table 2-53
MSR_DRAM_POWER_INFO	
06_5CH, 06_7AH	See Table 2-12

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H.	See Table 2-53
MSR_DRAM_POWER_LIMIT	
06_5CH, 06_7AH	See Table 2-12
06_2DH	See Table 2-23
06_3EH, 06_3FH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_6AH, 06_6CH	See Table 2-51
06_57H, 06_85H	See Table 2-53
MSR_EBC_FREQUENCY_ID	
0FH	See Table 2-55
MSR_EBC_HARD_POWERON	
0FH	See Table 2-55
MSR_EBC_SOFT_POWERON	
0FH	See Table 2-55
MSR_EBL_CR_POWERON	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.	See Table 2-6
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_EFSB_DRDY0	
0F_03H, 0F_04H	See Table 2-56
MSR_EFSB_DRDY1	
0F_03H, 0F_04H	See Table 2-56
MSR_EMON_L3_CTR_CTL0	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL1	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL2	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL3	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_EMON_L3_CTR_CTL4	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL5	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL6	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_CTR_CTL7	
06_0FH, 06_17H	See Table 2-3
0F_06H	See Table 2-57
MSR_EMON_L3_GL_CTL	
06_0FH, 06_17H	See Table 2-3
MSR_ERROR_CONTROL	
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
MSR_FAST_UNCORE_MSRS_CAPABILITY	
06_7DH, 06_7EH	See Table 2-44
MSR_FAST_UNCORE_MSRS_CTL	
06_7DH, 06_7EH	See Table 2-44
MSR_FAST_UNCORE_MSRS_STATUS	
06_7DH, 06_7EH	See Table 2-44
MSR_FEATURE_CONFIG	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH.	See Table 2-6
06_25H, 06_2CH.	See Table 2-18
06_2FH	See Table 2-19
06_2AH, 06_2DH.	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_FIRM_ESCR0	
0FH	See Table 2-55
MSR_FIRM_ESCR1	
0FH	See Table 2-55
MSR_FLAME_CCCR0	
0FH	See Table 2-55
MSR_FLAME_CCCR1	
0FH	See Table 2-55
MSR_FLAME_CCCR2	
0FH	See Table 2-55
MSR_FLAME_CCCR3	
0FH	See Table 2-55

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_FLAME_COUNTER0	
0FH	See Table 2-55
MSR_FLAME_COUNTER1	
0FH	See Table 2-55
MSR_FLAME_COUNTER2	
0FH	See Table 2-55
MSR_FLAME_COUNTER3	
0FH	See Table 2-55
MSR_FLAME_ESCR0	
0FH	See Table 2-55
MSR_FLAME_ESCR1	
0FH	See Table 2-55
MSR_FSB_ESCR0	
0FH	See Table 2-55
MSR_FSB_ESCR1	
0FH	See Table 2-55
MSR_FSB_FREQ	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_4CH	See Table 2-11
06_0EH	See Table 2-58
MSR_GQ_SNOOP_MESF	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_GRAPHICS_PERF_LIMIT_REASONS	
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_IFSB_BUSQ0	
0F_03H, 0F_04H	See Table 2-56
MSR_IFSB_BUSQ1	
0F_03H, 0F_04H	See Table 2-56
MSR_IFSB_CNTR7	
0F_03H, 0F_04H	See Table 2-56
MSR_IFSB_CTL6	
0F_03H, 0F_04H	See Table 2-56
MSR_IFSB_SNPQ0	
0F_03H, 0F_04H	See Table 2-56
MSR_IFSB_SNPQ1	
0F_03H, 0F_04H	See Table 2-56
MSR_IQ_CCCR0	
0FH	See Table 2-55
MSR_IQ_CCCR1	
0FH	See Table 2-55

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_IQ_CCCR2	
0FH	See Table 2-55
MSR_IQ_CCCR3	
0FH	See Table 2-55
MSR_IQ_CCCR4	
0FH	See Table 2-55
MSR_IQ_CCCR5	
0FH	See Table 2-55
MSR_IQ_COUNTER0	
0FH	See Table 2-55
MSR_IQ_COUNTER1	
0FH	See Table 2-55
MSR_IQ_COUNTER2	
0FH	See Table 2-55
MSR_IQ_COUNTER3	
0FH	See Table 2-55
MSR_IQ_COUNTER4	
0FH	See Table 2-55
MSR_IQ_COUNTER5	
0FH	See Table 2-55
MSR_IQ_ESCR0	
0FH	See Table 2-55
MSR_IQ_ESCR1	
0FH	See Table 2-55
MSR_IS_ESCR0	
0FH	See Table 2-55
MSR_IS_ESCR1	
0FH	See Table 2-55
MSR_ITLB_ESCR0	
0FH	See Table 2-55
MSR_ITLB_ESCR1	
0FH	See Table 2-55
MSR_IX_ESCR0	
0FH	See Table 2-55
MSR_IX_ESCR1	
0FH	See Table 2-55
MSR_LASTBRANCH_0	
0FH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_0_FROM_IP	
06_0FH, 06_17H	See Table 2-3

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH	See Table 2-12
06_7AH	See Table 2-13
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_0_TO_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH	See Table 2-12
06_7AH	See Table 2-13
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_1_FROM_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_1_TO_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_10_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_10_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_11_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_11_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_12_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_12_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_13_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_13_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_14_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_14_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_15_FROM_IP	
06_5CH, 06_7AH	See Table 2-12

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_15_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_16_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_16_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_17_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_17_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_18_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_18_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_19_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_19_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_2	
0FH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_2_FROM_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_2_TO_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_20_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_20_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_21_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_21_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_22_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_22_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_23_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_23_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_24_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_24_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_25_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_25_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_26_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_26_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_27_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_27_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_28_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_28_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_29_FROM_IP	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_29_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_3	
0FH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_3_FROM_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_3_TO_IP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_30_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_30_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_31_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_LASTBRANCH_31_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_LASTBRANCH_4	
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_4_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_4_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_5	
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_5_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_5_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_6	
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_6_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_6_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_7	
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_7_FROM_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_7_TO_IP	
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_8_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_8_TO_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_9_FROM_IP	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_9_TO_IP	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
0FH	See Table 2-55
MSR_LASTBRANCH_TOS	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_57H, 06_85H	See Table 2-53
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LASTBRANCH_INFO_0	
06_7AH	See Table 2-13
MSR_LBR_INFO_1	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_10	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_11	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_12	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_13	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_14	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_15	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_16	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_17	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_18	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_19	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_2	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_20	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_21	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_22	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_23	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_24	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_25	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_26	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_27	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_28	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_29	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_3	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_30	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_31	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_4	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_5	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_6	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_7	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_8	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_INFO_9	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
06_7AH	See Table 2-13
MSR_LBR_SELECT	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_LER_FROM_LIP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_57H, 06_85H	See Table 2-53
0FH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_LER_TO_LIP	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_57H, 06_85H	See Table 2-53
0FH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_M0_PMON_ADDR_MASK	
06_2EH	See Table 2-17
MSR_M0_PMON_ADDR_MATCH	
06_2EH	See Table 2-17
MSR_M0_PMON_BOX_CTRL	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
MSR_M0_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_M0_PMON_BOX_STATUS	
06_2EH	See Table 2-17
MSR_M0_PMON_CTR0	
06_2EH	See Table 2-17
MSR_M0_PMON_CTR1	
06_2EH	See Table 2-17
MSR_M0_PMON_CTR2	
06_2EH	See Table 2-17
MSR_M0_PMON_CTR3	
06_2EH	See Table 2-17
MSR_M0_PMON_CTR4	
06_2EH	See Table 2-17
MSR_M0_PMON_CTR5	
06_2EH	See Table 2-17
MSR_M0_PMON_DSP	
06_2EH	See Table 2-17
MSR_M0_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
MSR_M0_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
MSR_M0_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
MSR_M0_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
MSR_M0_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_M0_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_M0_PMON_ISS	
06_2EH	See Table 2-17
MSR_M0_PMON_MAP	
06_2EH	See Table 2-17
MSR_M0_PMON_MM_CONFIG	
06_2EH	See Table 2-17
MSR_M0_PMON_MSC_THR	
06_2EH	See Table 2-17
MSR_M0_PMON_PGT	
06_2EH	See Table 2-17
MSR_M0_PMON_PLD	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
MSR_M0_PMON_TIMESTAMP	
06_2EH	See Table 2-17
MSR_M0_PMON_ZDP	
06_2EH	See Table 2-17
MSR_M1_PMON_ADDR_MASK	
06_2EH	See Table 2-17
MSR_M1_PMON_ADDR_MATCH	
06_2EH	See Table 2-17
MSR_M1_PMON_BOX_CTRL	
06_2EH	See Table 2-17
MSR_M1_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_M1_PMON_BOX_STATUS	
06_2EH	See Table 2-17
MSR_M1_PMON_CTR0	
06_2EH	See Table 2-17
MSR_M1_PMON_CTR1	
06_2EH	See Table 2-17
MSR_M1_PMON_CTR2	
06_2EH	See Table 2-17
MSR_M1_PMON_CTR3	
06_2EH	See Table 2-17
MSR_M1_PMON_CTR4	
06_2EH	See Table 2-17
MSR_M1_PMON_CTR5	
06_2EH	See Table 2-17
MSR_M1_PMON_DSP	
06_2EH	See Table 2-17
MSR_M1_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
MSR_M1_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
MSR_M1_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
MSR_M1_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
MSR_M1_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_M1_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_M1_PMON_ISS	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
MSR_M1_PMON_MAP	
06_2EH	See Table 2-17
MSR_M1_PMON_MM_CONFIG	
06_2EH	See Table 2-17
MSR_M1_PMON_MSC_THR	
06_2EH	See Table 2-17
MSR_M1_PMON_PGT	
06_2EH	See Table 2-17
MSR_M1_PMON_PLD	
06_2EH	See Table 2-17
MSR_M1_PMON_TIMESTAMP	
06_2EH	See Table 2-17
MSR_M1_PMON_ZDP	
06_2EH	See Table 2-17
IA32_MC0_MISC / MSR_MC0_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
MSR_MC0_RESIDENCY	
06_57H, 06_85H	See Table 2-53
IA32_MC1_MISC / MSR_MC1_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
IA32_MC10_ADDR / MSR_MC10_ADDR	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC10_CTL / MSR_MC10_CTL	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC10_MISC / MSR_MC10_MISC	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38

MSR Name and CPUID DisplayFamily_DisplayModel	Location
IA32_MC10_STATUS / MSR_MC10_STATUS	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC11_ADDR / MSR_MC11_ADDR	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC11_CTL / MSR_MC11_CTL	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC11_MISC / MSR_MC11_MISC	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC11_STATUS / MSR_MC11_STATUS	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC12_ADDR / MSR_MC12_ADDR	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC12_CTL / MSR_MC12_CTL	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH	See Table 2-38
IA32_MC12_MISC / MSR_MC12_MISC	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC12_STATUS / MSR_MC12_STATUS	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC13_ADDR / MSR_MC13_ADDR	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC13_CTL / MSR_MC13_CTL	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC13_MISC / MSR_MC13_MISC	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC13_STATUS / MSR_MC13_STATUS	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC14_ADDR / MSR_MC14_ADDR	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH	See Table 2-38
IA32_MC14_CTL / MSR_MC14_CTL	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC14_MISC / MSR_MC14_MISC	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC14_STATUS / MSR_MC14_STATUS	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC15_ADDR / MSR_MC15_ADDR	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC15_CTL / MSR_MC15_CTL	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC15_MISC / MSR_MC15_MISC	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC15_STATUS / MSR_MC15_STATUS	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH	See Table 2-38
IA32_MC16_ADDR / MSR_MC16_ADDR	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC16_CTL / MSR_MC16_CTL	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC16_MISC / MSR_MC16_MISC	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC16_STATUS / MSR_MC16_STATUS	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC17_ADDR / MSR_MC17_ADDR	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC17_CTL / MSR_MC17_CTL	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC17_MISC / MSR_MC17_MISC	
06_2EH	See Table 2-17
06_2DH	See Table 2-23

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC17_STATUS / MSR_MC17_STATUS	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC18_ADDR / MSR_MC18_ADDR	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC18_CTL / MSR_MC18_CTL	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC18_MISC / MSR_MC18_MISC	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC18_STATUS / MSR_MC18_STATUS	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC19_ADDR / MSR_MC19_ADDR	
06_2EH	See Table 2-17
06_2DH	See Table 2-23

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC19_CTL / MSR_MC19_CTL	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC19_MISC / MSR_MC19_MISC	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC19_STATUS / MSR_MC19_STATUS	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC2_MISC / MSR_MC2_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
IA32_MC20_ADDR / MSR_MC20_ADDR	
06_2EH	See Table 2-17
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC20_CTL / MSR_MC20_CTL	
06_2EH	See Table 2-17
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC20_MISC / MSR_MC20_MISC	
06_2EH	See Table 2-17
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38

MSR Name and CPUID DisplayFamily_DisplayModel	Location
IA32_MC20_STATUS / MSR_MC20_STATUS	
06_2EH	See Table 2-17
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC21_ADDR / MSR_MC21_ADDR	
06_2EH	See Table 2-17
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4F	See Table 2-38
IA32_MC21_CTL / MSR_MC21_CTL	
06_2EH	See Table 2-17
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4F	See Table 2-38
IA32_MC21_MISC / MSR_MC21_MISC	
06_2EH	See Table 2-17
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4F	See Table 2-38
IA32_MC21_STATUS / MSR_MC21_STATUS	
06_2EH	See Table 2-17
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4F	See Table 2-38
IA32_MC22_ADDR / MSR_MC22_ADDR	
06_3EH	See Table 2-26
IA32_MC22_CTL / MSR_MC22_CTL	
06_3EH	See Table 2-26
IA32_MC22_MISC / MSR_MC22_MISC	
06_3EH	See Table 2-26
IA32_MC22_STATUS / MSR_MC22_STATUS	
06_3EH	See Table 2-26
IA32_MC23_ADDR / MSR_MC23_ADDR	
06_3EH	See Table 2-26
IA32_MC23_CTL / MSR_MC23_CTL	
06_3EH	See Table 2-26
IA32_MC23_MISC / MSR_MC23_MISC	
06_3EH	See Table 2-26
IA32_MC23_STATUS / MSR_MC23_STATUS	
06_3EH	See Table 2-26
IA32_MC24_ADDR / MSR_MC24_ADDR	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH	See Table 2-26
IA32_MC24_CTL / MSR_MC24_CTL	
06_3EH	See Table 2-26
IA32_MC24_MISC / MSR_MC24_MISC	
06_3EH	See Table 2-26
IA32_MC24_STATUS / MSR_MC24_STATUS	
06_3EH	See Table 2-26
IA32_MC25_ADDR / MSR_MC25_ADDR	
06_3EH	See Table 2-26
IA32_MC25_CTL / MSR_MC25_CTL	
06_3EH	See Table 2-26
IA32_MC25_MISC / MSR_MC25_MISC	
06_3EH	See Table 2-26
IA32_MC25_STATUS / MSR_MC25_STATUS	
06_3EH	See Table 2-26
IA32_MC26_ADDR / MSR_MC26_ADDR	
06_3EH	See Table 2-26
IA32_MC26_CTL / MSR_MC26_CTL	
06_3EH	See Table 2-26
IA32_MC26_MISC / MSR_MC26_MISC	
06_3EH	See Table 2-26
IA32_MC26_STATUS / MSR_MC26_STATUS	
06_3EH	See Table 2-26
IA32_MC27_ADDR / MSR_MC27_ADDR	
06_3EH	See Table 2-26
IA32_MC27_CTL / MSR_MC27_CTL	
06_3EH	See Table 2-26
IA32_MC27_MISC / MSR_MC27_MISC	
06_3EH	See Table 2-26
IA32_MC27_STATUS / MSR_MC27_STATUS	
06_3EH	See Table 2-26
IA32_MC28_ADDR / MSR_MC28_ADDR	
06_3EH	See Table 2-26
IA32_MC28_CTL / MSR_MC28_CTL	
06_3EH	See Table 2-26
IA32_MC28_MISC / MSR_MC28_MISC	
06_3EH	See Table 2-26
IA32_MC28_STATUS / MSR_MC28_STATUS	
06_3EH	See Table 2-26
IA32_MC29_ADDR / MSR_MC29_ADDR	
06_3EH	See Table 2-27
IA32_MC29_CTL / MSR_MC29_CTL	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH	See Table 2-27
IA32_MC29_MISC / MSR_MC29_MISC	
06_3EH	See Table 2-27
IA32_MC29_STATUS / MSR_MC29_STATUS	
06_3EH	See Table 2-27
IA32_MC3_ADDR / MSR_MC3_ADDR	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H	See Table 2-53
06_0EH	See Table 2-58
06_09H	See Table 2-59
IA32_MC3_CTL / MSR_MC3_CTL	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H	See Table 2-53
06_0EH	See Table 2-58
06_09H	See Table 2-59
IA32_MC3_MISC / MSR_MC3_MISC	
06_0FH, 06_17H	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_0EH	See Table 2-58
IA32_MC3_STATUS / MSR_MC3_STATUS	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H	See Table 2-53
06_0EH	See Table 2-58
06_09H	See Table 2-59
IA32_MC30_ADDR / MSR_MC30_ADDR	
06_3EH	See Table 2-27
IA32_MC30_CTL / MSR_MC30_CTL	
06_3EH	See Table 2-27
IA32_MC30_MISC / MSR_MC30_MISC	
06_3EH	See Table 2-27
IA32_MC30_STATUS / MSR_MC30_STATUS	
06_3EH	See Table 2-27
IA32_MC31_ADDR / MSR_MC31_ADDR	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH	See Table 2-27
IA32_MC31_CTL / MSR_MC31_CTL	
06_3EH	See Table 2-27
IA32_MC31_MISC / MSR_MC31_MISC	
06_3EH	See Table 2-27
IA32_MC31_STATUS / MSR_MC31_STATUS	
06_3EH	See Table 2-27
IA32_MC4_ADDR / MSR_MC4_ADDR	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H	See Table 2-53
06_0EH	See Table 2-58
06_09H	See Table 2-59
IA32_MC4_CTL / MSR_MC4_CTL	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H	See Table 2-53
06_0EH	See Table 2-58
06_09H	See Table 2-59
IA32_MC4_CTL2 / MSR_MC4_CTL2	
06_2AH, 06_2DH	See Table 2-20
IA32_MC4_STATUS / MSR_MC4_STATUS	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_57H, 06_85H	See Table 2-53
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_MC5_ADDR / MSR_MC5_ADDR	
06_0FH, 06_17H	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3FH	See Table 2-32
06_4FH	See Table 2-38
06_57H, 06_85H	See Table 2-53

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_0EH	See Table 2-58
IA32_MC5_CTL / MSR_MC5_CTL	
06_0FH, 06_17H	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3FH	See Table 2-32
06_4FH	See Table 2-38
06_57H, 06_85H	See Table 2-53
06_0EH	See Table 2-58
IA32_MC5_MISC / MSR_MC5_MISC	
06_0FH, 06_17H	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3FH	See Table 2-32
06_4FH	See Table 2-38
06_0EH	See Table 2-58
IA32_MC5_STATUS / MSR_MC5_STATUS	
06_0FH, 06_17H	See Table 2-3
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3FH	See Table 2-32
06_4FH	See Table 2-38
06_57H, 06_85H	See Table 2-53
06_0EH	See Table 2-58
IA32_MC6_ADDR / MSR_MC6_ADDR	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC6_CTL / MSR_MC6_CTL	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4FH	See Table 2-38
MSR_MC6_DEMOTION_POLICY_CONFIG	
06_37H	See Table 2-9
IA32_MC6_MISC / MSR_MC6_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
MSR_MC6_RESIDENCY_COUNTER	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_37H	See Table 2-9
06_57H, 06_85H	See Table 2-53
IA32_CORE_CAPABILITIES (Note there are no architecturally defined bits; all bits are model-specific)	
06_86H	See Table 2-14
06_8CH, 06_8DH	See Table 2-45
IA32_MC6_STATUS / MSR_MC6_STATUS	
06_0FH, 06_17H	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3FH	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC7_ADDR / MSR_MC7_ADDR	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC7_CTL / MSR_MC7_CTL	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC7_MISC / MSR_MC7_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC7_STATUS / MSR_MC7_STATUS	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC8_ADDR / MSR_MC8_ADDR	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC8_CTL / MSR_MC8_CTL	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC8_MISC / MSR_MC8_MISC	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC8_STATUS / MSR_MC8_STATUS	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_4FH	See Table 2-38
IA32_MC9_ADDR / MSR_MC9_ADDR	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38

MSR Name and CPUID DisplayFamily_DisplayModel	Location
IA32_MC9_CTL / MSR_MC9_CTL	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC9_MISC / MSR_MC9_MISC	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
IA32_MC9_STATUS / MSR_MC9_STATUS	
06_2EH	See Table 2-17
06_2DH	See Table 2-23
06_3EH	See Table 2-26
06_3F	See Table 2-32
06_56H, 06_4FH	See Table 2-37
06_4FH	See Table 2-38
MSR_MCG_MISC	
0FH	See Table 2-55
MSR_MCG_R10	
0FH	See Table 2-55
MSR_MCG_R11	
0FH	See Table 2-55
MSR_MCG_R12	
0FH	See Table 2-55
MSR_MCG_R13	
0FH	See Table 2-55
MSR_MCG_R14	
0FH	See Table 2-55
MSR_MCG_R15	
0FH	See Table 2-55
MSR_MCG_R8	
0FH	See Table 2-55
MSR_MCG_R9	
0FH	See Table 2-55
MSR_MCG_RAX	
0FH	See Table 2-55
MSR_MCG_RBP	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
0FH	See Table 2-55
MSR_MS_COUNTER0	
0FH	See Table 2-55
MSR_MS_COUNTER1	
0FH	See Table 2-55
MSR_MS_COUNTER2	
0FH	See Table 2-55
MSR_MS_COUNTER3	
0FH	See Table 2-55
MSR_MS_ESCR0	
0FH	See Table 2-55
MSR_MS_ESCR1	
0FH	See Table 2-55
MSR_MTRRCAP	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH	See Table 2-39
MSR_OFFCORE_RSP_0	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_OFFCORE_RSP_1	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_25H, 06_2CH	See Table 2-18
06_2FH	See Table 2-19
06_2AH, 06_2DH	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PACKAGE_ENERGY_TIME_STATUS	
06_6AH, 06_6CH	See Table 2-51
MSR_PCIE_PLL_RATIO	
06_3FH	See Table 2-32
MSR_PCU_PMON_BOX_CTL	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_PCU_PMON_BOX_FILTER	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_PCU_PMON_BOX_STATUS	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_PCU_PMON_CTR0	
06_2DH	See Table 2-24
06_3FH	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_PCU_PMON_CTR1	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_PCU_PMON_CTR2	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_PCU_PMON_CTR3	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_PCU_PMON_EVNTSEL0	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_PCU_PMON_EVNTSEL1	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_PCU_PMON_EVNTSEL2	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_PCU_PMON_EVNTSEL3	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_PEBS_DATA_CFG	
06_7DH, 06_7EH	See Table 2-44
MSR_PEBS_ENABLE	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH	See Table 2-12
06_7AH	See Table 2-13
06_86H	See Table 2-14
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3EH	See Table 2-27
06_57H, 06_85H	See Table 2-53
0FH	See Table 2-55
MSR_PEBS_FRONTEND	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH	See Table 2-39
MSR_PEBS_LD_LAT	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
MSR_PEBS_MATRIX_VERT	
0FH	See Table 2-55

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_PEBS_NUM_ALT	
06_2DH	See Table 2-23
MSR_PERF_CAPABILITIES	
06_0FH, 06_17H	See Table 2-3
MSR_PERF_GLOBAL_CTRL	
06_0FH, 06_17H	See Table 2-3
MSR_PERF_GLOBAL_OVF_CTRL	
06_0FH, 06_17H	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
MSR_PERF_GLOBAL_STATUS	
06_0FH, 06_17H	See Table 2-3
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
MSR_PERF_METRICS	
06_7DH, 06_7EH	See Table 2-44
MSR_PERF_STATUS	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_2AH, 06_2DH	See Table 2-20
MSR_PKG_C10_RESIDENCY	
06_5CH, 06_7AH	See Table 2-12
06_45H	See Table 2-30 and Table 2-31
06_4FH	See Table 2-38
MSR_PKG_C2_RESIDENCY	
06_27H	See Table 2-5
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PKG_C3_RESIDENCY	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_66H	See Table 2-42
06_57H, 06_85H	See Table 2-53
MSR_PKG_C4_RESIDENCY	
06_27H	See Table 2-5
MSR_PKG_C6_RESIDENCY	
06_27H	See Table 2-5
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_57H, 06_85H	See Table 2-53
MSR_PKG_C7_RESIDENCY	
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	See Table 2-15
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PKG_C8_RESIDENCY	
06_45H	See Table 2-31
06_4FH	See Table 2-38
MSR_PKG_C9_RESIDENCY	
06_45H	See Table 2-31
06_4FH	See Table 2-38
MSR_PKG_CST_CONFIG_CONTROL	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_4CH	See Table 2-11
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3AH	See Table 2-25
06_3EH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-30
06_45H	See Table 2-31
06_3F	See Table 2-32
06_3DH	See Table 2-35
06_56H, 06_4FH	See Table 2-36
06_57H, 06_85H	See Table 2-53
MSR_PKG_ENERGY_STATUS	
06_37H, 06_4AH, 06_4CH, 06_5AH, 06_5DH	See Table 2-8
06_5CH, 06_7AH, 06_86H	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3DH, 06_3EH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PKG_HDC_CONFIG	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PKG_HDC_DEEP_RESIDENCY	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PKG_HDC_SHALLOW_RESIDENCY	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PKG_PERF_STATUS	
06_5CH, 06_7AH, 06_86H	See Table 2-12
06_2DH	See Table 2-23

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3AH, 06_3EH	See Table 2-26
06_3CH, 06_3DH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-29
06_57H, 06_85H	See Table 2-53
MSR_PKG_POWER_INFO	
06_4DH	See Table 2-10
06_5CH, 06_7AH, 06_86H	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3DH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PKG_POWER_LIMIT	
06_37H, 06_4AH, 06_4CH, 06_5AH, 06_5DH	See Table 2-8
06_4DH	See Table 2-10
06_5CH, 06_7AH, 06_86H	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3DH, 06_3EH, 06_3FH, 06_45H, 06_46H, 06_47H, 06_4EH, 06_4FH, 06_55H, 06_56H, 06_5EH, 06_66H, 06_8EH, 06_9EH, 06_7DH, 06_7EH	See Table 2-20
06_57H, 06_85H	See Table 2-53
MSR_PKGC_IRTL1	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H.	See Table 2-29
MSR_PKGC_IRTL2	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H.	See Table 2-29
MSR_PKGC3_IRTL	
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_2DH	See Table 2-20
MSR_PKGC6_IRTL	
06_2AH, 06_2DH	See Table 2-20
MSR_PKGC7_IRTL	
06_2AH	See Table 2-21
MSR_PLATFORM_BRV	
0FH	See Table 2-55
MSR_PLATFORM_ENERGY_COUNTER	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PLATFORM_ID	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH	See Table 2-7
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
MSR_PLATFORM_INFO	
06_5CH, 06_7AH	See Table 2-12

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3AH	See Table 2-25
06_3EH	See Table 2-26
06_3CH, 06_45H, 06_46H	See Table 2-29 and Table 2-30
06_56H, 06_4FH	See Table 2-36
06_57H	See Table 2-53
MSR_PLATFORM_POWER_LIMIT	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_PMG_IO_CAPTURE_BASE	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_4CH	See Table 2-11
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3AH	See Table 2-25
06_3EH	See Table 2-26
06_57H	See Table 2-53
MSR_PMH_ESCR0	
0FH	See Table 2-55
MSR_PMH_ESCR1	
0FH	See Table 2-55
MSR_PMON_GLOBAL_CONFIG	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_PMON_GLOBAL_CTL	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_PMON_GLOBAL_STATUS	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_POWER_CTL	
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
MSR_PP0_ENERGY_STATUS	
06_37H, 06_4AH, 06_5AH, 06_5DH	See Table 2-8
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H	See Table 2-53
MSR_PP0_POLICY	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2AH, 06_45H	See Table 2-21
MSR_PP0_POWER_LIMIT	
06_4CH	See Table 2-11
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_57H	See Table 2-53
MSR_PP1_ENERGY_STATUS	
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_45H	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_PP1_POLICY	
06_2AH, 06_45H	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_PP1_POWER_LIMIT	
06_2AH, 06_45H	See Table 2-21
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_PPERF	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
IA32_PPIN / MSR_PPIN	
06_3EH	See Table 2-26
06_56H, 06_4FH	See Table 2-36
06_55H	See Table 2-50
06_57H, 06_85H	See Table 2-53
IA32_PPIN_CTL / MSR_PPIN_CTL	
06_3EH	See Table 2-26
06_56H, 06_4FH	See Table 2-36
06_55H	See Table 2-50
06_57H, 06_85H	See Table 2-53
MSR_PRMRR_BASE_0	
06_7DH, 06_7EH	See Table 2-44
MSR_PRMRR_PHYS_BASE	
06_8EH, 06_9EH	See Table 2-41
MSR_PRMRR_PHYS_MASK	
06_8EH, 06_9EH	See Table 2-41
MSR_PRMRR_VALID_CONFIG	
06_8EH, 06_9EH	See Table 2-41
MSR_RELOAD_FIXED_CTRx	
06_86H	See Table 2-14
MSR_RELOAD_PMCx	
06_86H	See Table 2-14
MSR_RING_RATIO_LIMIT	
06_8EH, 06_9EH	See Table 2-41

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_R0_PMON_BOX_CTRL	
06_2EH	See Table 2-17
MSR_R0_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_R0_PMON_BOX_STATUS	
06_2EH	See Table 2-17
MSR_R0_PMON_CTR0	
06_2EH	See Table 2-17
MSR_R0_PMON_CTR1	
06_2EH	See Table 2-17
MSR_R0_PMON_CTR2	
06_2EH	See Table 2-17
MSR_R0_PMON_CTR3	
06_2EH	See Table 2-17
MSR_R0_PMON_CTR4	
06_2EH	See Table 2-17
MSR_R0_PMON_CTR5	
06_2EH	See Table 2-17
MSR_R0_PMON_CTR6	
06_2EH	See Table 2-17
MSR_R0_PMON_CTR7	
06_2EH	See Table 2-17
MSR_R0_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
MSR_R0_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
MSR_R0_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
MSR_R0_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
MSR_R0_PMON_EVNT_SEL4	
06_2EH	See Table 2-17
MSR_R0_PMON_EVNT_SEL5	
06_2EH	See Table 2-17
MSR_R0_PMON_EVNT_SEL6	
06_2EH	See Table 2-17
MSR_R0_PMON_EVNT_SEL7	
06_2EH	See Table 2-17
MSR_R0_PMON_IPERF0_P0	
06_2EH	See Table 2-17
MSR_R0_PMON_IPERF0_P1	
06_2EH	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_R0_PMON_IPERF0_P2	
06_2EH	See Table 2-17
MSR_R0_PMON_IPERF0_P3	
06_2EH	See Table 2-17
MSR_R0_PMON_IPERF0_P4	
06_2EH	See Table 2-17
MSR_R0_PMON_IPERF0_P5	
06_2EH	See Table 2-17
MSR_R0_PMON_IPERF0_P6	
06_2EH	See Table 2-17
MSR_R0_PMON_IPERF0_P7	
06_2EH	See Table 2-17
MSR_R0_PMON_QLX_P0	
06_2EH	See Table 2-17
MSR_R0_PMON_QLX_P1	
06_2EH	See Table 2-17
MSR_R0_PMON_QLX_P2	
06_2EH	See Table 2-17
MSR_R0_PMON_QLX_P3	
06_2EH	See Table 2-17
MSR_R1_PMON_BOX_CTRL	
06_2EH	See Table 2-17
MSR_R1_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_R1_PMON_BOX_STATUS	
06_2EH	See Table 2-17
MSR_R1_PMON_CTR10	
06_2EH	See Table 2-17
MSR_R1_PMON_CTR11	
06_2EH	See Table 2-17
MSR_R1_PMON_CTR12	
06_2EH	See Table 2-17
MSR_R1_PMON_CTR13	
06_2EH	See Table 2-17
MSR_R1_PMON_CTR14	
06_2EH	See Table 2-17
MSR_R1_PMON_CTR15	
06_2EH	See Table 2-17
MSR_R1_PMON_CTR8	
06_2EH	See Table 2-17
MSR_R1_PMON_CTR9	
06_2EH	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_R1_PMON_EVNT_SEL10	
06_2EH	See Table 2-17
MSR_R1_PMON_EVNT_SEL11	
06_2EH	See Table 2-17
MSR_R1_PMON_EVNT_SEL12	
06_2EH	See Table 2-17
MSR_R1_PMON_EVNT_SEL13	
06_2EH	See Table 2-17
MSR_R1_PMON_EVNT_SEL14	
06_2EH	See Table 2-17
MSR_R1_PMON_EVNT_SEL15	
06_2EH	See Table 2-17
MSR_R1_PMON_EVNT_SEL8	
06_2EH	See Table 2-17
MSR_R1_PMON_EVNT_SEL9	
06_2EH	See Table 2-17
MSR_R1_PMON_IPERF1_P10	
06_2EH	See Table 2-17
MSR_R1_PMON_IPERF1_P11	
06_2EH	See Table 2-17
MSR_R1_PMON_IPERF1_P12	
06_2EH	See Table 2-17
MSR_R1_PMON_IPERF1_P13	
06_2EH	See Table 2-17
MSR_R1_PMON_IPERF1_P14	
06_2EH	See Table 2-17
MSR_R1_PMON_IPERF1_P15	
06_2EH	See Table 2-17
MSR_R1_PMON_IPERF1_P8	
06_2EH	See Table 2-17
MSR_R1_PMON_IPERF1_P9	
06_2EH	See Table 2-17
MSR_R1_PMON_QLX_P4	
06_2EH	See Table 2-17
MSR_R1_PMON_QLX_P5	
06_2EH	See Table 2-17
MSR_R1_PMON_QLX_P6	
06_2EH	See Table 2-17
MSR_R1_PMON_QLX_P7	
06_2EH	See Table 2-17
MSR_RAPL_POWER_UNIT	
06_37H, 06_4AH, 06_5AH, 06_5DH	See Table 2-8

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_4DH	See Table 2-10
06_5CH, 06_7AH	See Table 2-12
06_2AH, 06_2DH, 06_3AH, 06_3CH, 06_3EH, 06_3FH, 06_45H, 06_46H	See Table 2-20
06_3FH	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_57H	See Table 2-53
MSR_RAT_ESCR0	
0FH	See Table 2-55
MSR_RAT_ESCR1	
0FH	See Table 2-55
MSR_RING_PERF_LIMIT_REASONS	
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_S0_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S0_PMON_BOX_FILTER	
06_3FH	See Table 2-33
MSR_S0_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_S0_PMON_BOX_STATUS	
06_2EH	See Table 2-17
MSR_S0_PMON_CTR0	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S0_PMON_CTR1	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S0_PMON_CTR2	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S0_PMON_CTR3	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S0_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S0_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S0_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_3FH	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_S0_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S0_PMON_MASK	
06_2EH	See Table 2-17
MSR_S0_PMON_MATCH	
06_2EH	See Table 2-17
MSR_S1_PMON_BOX_CTRL	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S1_PMON_BOX_FILTER	
06_3FH	See Table 2-33
MSR_S1_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_S1_PMON_BOX_STATUS	
06_2EH	See Table 2-17
MSR_S1_PMON_CTR0	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S1_PMON_CTR1	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S1_PMON_CTR2	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S1_PMON_CTR3	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S1_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S1_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S1_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S1_PMON_EVNT_SEL3	
06_2EH	See Table 2-17
06_3FH	See Table 2-33
MSR_S1_PMON_MASK	
06_2EH	See Table 2-17

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_S1_PMON_MATCH	
06_2EH	See Table 2-17
MSR_S2_PMON_BOX_CTL	
06_3FH	See Table 2-33
MSR_S2_PMON_BOX_FILTER	
06_3FH	See Table 2-33
MSR_S2_PMON_CTR0	
06_3FH	See Table 2-33
MSR_S2_PMON_CTR1	
06_3FH	See Table 2-33
MSR_S2_PMON_CTR2	
06_3FH	See Table 2-33
MSR_S2_PMON_CTR3	
06_3FH	See Table 2-33
MSR_S2_PMON_EVNTSEL0	
06_3FH	See Table 2-33
MSR_S2_PMON_EVNTSEL1	
06_3FH	See Table 2-33
MSR_S2_PMON_EVNTSEL2	
06_3FH	See Table 2-33
MSR_S2_PMON_EVNTSEL3	
06_3FH	See Table 2-33
MSR_S3_PMON_BOX_CTL	
06_3FH	See Table 2-33
MSR_S3_PMON_BOX_FILTER	
06_3FH	See Table 2-33
MSR_S3_PMON_CTR0	
06_3FH	See Table 2-33
MSR_S3_PMON_CTR1	
06_3FH	See Table 2-33
MSR_S3_PMON_CTR2	
06_3FH	See Table 2-33
MSR_S3_PMON_CTR3	
06_3FH	See Table 2-33
MSR_S3_PMON_EVNTSEL0	
06_3FH	See Table 2-33
MSR_S3_PMON_EVNTSEL1	
06_3FH	See Table 2-33
MSR_S3_PMON_EVNTSEL2	
06_3FH	See Table 2-33
MSR_S3_PMON_EVNTSEL3	
06_3FH	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_SAAT_ESCR0	
0FH	See Table 2-55
MSR_SAAT_ESCR1	
0FH	See Table 2-55
MSR_SGXOWNEREPOCH0	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_SGXOWNEREPOCH1	
06_5CH, 06_7AH	See Table 2-12
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_SMI_COUNT	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_57H	See Table 2-53
MSR_SMM_BLOCKED	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_SMM_DELAYED	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_SMM_FEATURE_CONTROL	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H	See Table 2-30
MSR_SMM_MCA_CAP	
06_5CH, 06_7AH	See Table 2-12
06_3CH, 06_45H, 06_46H	See Table 2-30
06_3FH	See Table 2-32
06_56H, 06_4FH	See Table 2-36
06_57H	See Table 2-53
MSR_SMRR_PHYSBASE	
06_0FH, 06_17H	See Table 2-3
MSR_SMRR_PHYSMASK	
06_0FH, 06_17H	See Table 2-3
MSR_SSU_ESCR0	
0FH	See Table 2-55
MSR_TBPU_ESCR0	
0FH	See Table 2-55
MSR_TBPU_ESCR1	
0FH	See Table 2-55

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_TC_ESCR0	
0FH	See Table 2-55
MSR_TC_ESCR1	
0FH	See Table 2-55
MSR_TC_PRECISE_EVENT	
0FH	See Table 2-55
MSR_TEMPERATURE_TARGET	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
06_2AH, 06_2DH	See Table 2-20
06_3EH	See Table 2-26
06_56H, 06_4FH	See Table 2-36
06_57H	See Table 2-53
MSR_TEST_CTRL	
P6 Family	See Table 2-60
MSR_THERM2_CTL	
06_0FH, 06_17H	See Table 2-3
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	See Table 2-4
0FH	See Table 2-55
06_0EH	See Table 2-58
06_09H	See Table 2-59
MSR_THREAD_ID_INFO	
06_3FH	See Table 2-32
MSR_TRACE_HUB_STH_ACPIBAR_BASE	
06_8EH, 06_9EH	See Table 2-41
MSR_TURBO_ACTIVATION_RATIO	
06_5CH, 06_7AH	See Table 2-12
06_3AH	See Table 2-25
06_3CH, 06_45H, 06_46H	See Table 2-29
06_57H	See Table 2-53
MSR_TURBO_GROUP_CORECNT	
06_5CH, 06_7AH	See Table 2-12
MSR_TURBO_POWER_CURRENT_LIMIT	
06_1AH, 06_1EH, 06_1FH, 06_2EH	See Table 2-15
MSR_TURBO_RATIO_LIMIT	
06_37H, 06_4AH, 06_4DH, 06_5AH, 06_5DH, 06_5CH, 06_7AH	See Table 2-6
06_4DH	See Table 2-10
06_5CH, 06_7AH	See Table 2-12
06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH	See Table 2-15
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
06_2EH	See Table 2-17
06_25H, 06_2CH	See Table 2-18

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2FH	See Table 2-19
06_2AH, 06_45H	See Table 2-21
06_2DH	See Table 2-23
06_3EH	See Table 2-26 and Table 2-27
06_3CH, 06_45H, 06_46H	See Table 2-30
06_3FH	See Table 2-32
06_3DH	See Table 2-35
06_56H, 06_4FH	See Table 2-36
06_55H	See Table 2-50
06_57H	See Table 2-53
MSR_TURBO_RATIO_LIMIT1	
06_3EH	See Table 2-26 and Table 2-27
06_3FH	See Table 2-32
06_56H, 06_4FH	See Table 2-36
MSR_TURBO_RATIO_LIMIT2	
06_3FH	See Table 2-32
MSR_TURBO_RATIO_LIMIT3	
06_56H	See Table 2-37
06_4FH	See Table 2-38
MSR_TURBO_RATIO_LIMIT_CORES	
06_55H	See Table 2-50
MSR_U_PMON_BOX_STATUS	
06_3EH	See Table 2-28
06_3FH	See Table 2-33
MSR_U_PMON_CTR	
06_2EH	See Table 2-17
MSR_U_PMON_CTR0	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_U_PMON_CTR1	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_U_PMON_EVNT_SEL	
06_2EH	See Table 2-17
MSR_U_PMON_EVNTSEL0	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_U_PMON_EVNTSEL1	
06_2DH	See Table 2-24
06_3FH	See Table 2-33

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_U_PMON_GLOBAL_CTRL	
06_2EH	See Table 2-17
MSR_U_PMON_GLOBAL_OVF_CTRL	
06_2EH	See Table 2-17
MSR_U_PMON_GLOBAL_STATUS	
06_2EH	See Table 2-17
MSR_U_PMON_UCLK_FIXED_CTL	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_U_PMON_UCLK_FIXED_CTR	
06_2DH	See Table 2-24
06_3FH	See Table 2-33
MSR_U2L_ESCR0	
0FH	See Table 2-55
MSR_U2L_ESCR1	
0FH	See Table 2-55
MSR_UNC_ARB_PERFCTR0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_ARB_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_ARB_PERFEVTSEL0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_ARB_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFCTR0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFCTR2	
06_2AH	See Table 2-22

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNC_CBO_0_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_0_PERFEVTSEL0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_0_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_0_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_0_UNIT_STATUS	
06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFCTR0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFCTR2	
06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFEVTSEL0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_1_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_1_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_1_UNIT_STATUS	
06_2AH	See Table 2-22

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNC_CBO_2_PERFCTR0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFCTR2	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_PERFEVTSEL0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_2_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_2_UNIT_STATUS	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFCTR0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_3_PERFCTR1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_3_PERFCTR2	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFEVTSEL0	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNC_CBO_3_PERFEVTSEL1	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_CBO_3_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_3_UNIT_STATUS	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR0	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR1	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR2	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFCTR3	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSEL0	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSEL1	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSEL2	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_PERFEVTSEL3	
06_2AH	See Table 2-22
MSR_UNC_CBO_4_UNIT_STATUS	
06_2AH	See Table 2-22
MSR_UNC_CBO_CONFIG	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_FIXED_CTR	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_FIXED_CTRL	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_GLOBAL_CTRL	
06_2AH	See Table 2-22

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNC_PERF_GLOBAL_STATUS	
06_2AH	See Table 2-22
06_3CH, 06_45H, 06_46H	See Table 2-30
06_4EH, 06_5EH	See Table 2-40
MSR_UNCORE_ADDR_OPCODE_MATCH	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_FIXED_CTR_CTRL	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_FIXED_CTR0	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERF_GLOBAL_CTRL	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERF_GLOBAL_STATUS	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL0	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL1	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL2	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL3	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL4	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL5	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL6	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PERFEVTSEL7	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC0	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC1	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC2	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC3	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16

MSR Name and CPUID DisplayFamily_DisplayModel	Location
MSR_UNCORE_PMC4	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC5	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
06_2EH	See Table 2-17
MSR_UNCORE_PMC6	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PMC7	
06_1AH, 06_1EH, 06_1FH, 06_25H, 06_2CH	See Table 2-16
MSR_UNCORE_PRMRR_BASE	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_UNCORE_PRMRR_MASK	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MSR_UNCORE_PRMRR_PHYS_BASE	
06_8EH, 06_9EH	See Table 2-41
MSR_UNCORE_PRMRR_PHYS_MASK	
06_8EH, 06_9EH	See Table 2-41
MSR_VR_CURRENT_CONFIG	
06_8CH, 06_8DH	See Table 2-45
MSR_W_PMON_BOX_CTRL	
06_2EH	See Table 2-17
MSR_W_PMON_BOX_OVF_CTRL	
06_2EH	See Table 2-17
MSR_W_PMON_BOX_STATUS	
06_2EH	See Table 2-17
MSR_W_PMON_CTR0	
06_2EH	See Table 2-17
MSR_W_PMON_CTR1	
06_2EH	See Table 2-17
MSR_W_PMON_CTR2	
06_2EH	See Table 2-17
MSR_W_PMON_CTR3	
06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SEL0	
06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SEL1	
06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SEL2	
06_2EH	See Table 2-17
MSR_W_PMON_EVNT_SEL3	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_2EH	See Table 2-17
MSR_W_PMON_FIXED_CTR	
06_2EH	See Table 2-17
MSR_W_PMON_FIXED_CTR_CTL	
06_2EH	See Table 2-17
MSR_WEIGHTED_CORE_C0	
06_4EH, 06_5EH, 06_55H, 06_8EH, 06_9EH, 06_66H, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_6CH	See Table 2-39
MTRRfix16K_80000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix16K_A0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_C0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_C8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_D0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_D8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_E0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_E8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_F0000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix4K_F8000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRfix64K_00000	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase0	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase1	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase2	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase3	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase4	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase5	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase6	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysBase7	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask0	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask1	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask2	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask3	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask4	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask5	
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask6	

MSR Name and CPUID DisplayFamily_DisplayModel	Location
06_0EH	See Table 2-58
P6 Family	See Table 2-60
MTRRphysMask7	
06_0EH	See Table 2-58
P6 Family	See Table 2-60